



SIEBEL[®] 7
eBusiness

SIEBEL TOOLS REFERENCE

VERSION 7.5.3, REV. A
OCTOBER 2003

Siebel Systems, Inc., 2207 Bridgepointe Parkway, San Mateo, CA 94404
Copyright © 2003 Siebel Systems, Inc.
All rights reserved.
Printed in the United States of America

No part of this publication may be stored in a retrieval system, transmitted, or reproduced in any way, including but not limited to photocopy, photographic, magnetic, or other record, without the prior agreement and written permission of Siebel Systems, Inc.

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.

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

Introduction

How This Guide Is Organized	25
Revision History	26

Chapter 1. Siebel Tools Installation

Preinstallation Tasks	31
Verifying Siebel Tools Prerequisites	31
Installing a Database Server or Sample Database	32
Installing Siebel Tools	32
Language Pack Installation	35
ODBC Installation	35
Postinstallation Tasks	37
Verifying Successful Installation	37
Verifying the Siebel Tools Directory Structure	38
Verify Read/Write Access to Tools Directories	40
Siebel Tools ODBC Data Sources	40
Running Multiple Local Databases	41
Repository Naming Conventions	43
Setting Up the Development Environment	44

Chapter 2. Siebel Architecture (Basic Concepts)

About Siebel Tools	49
Application Architecture Overview	50
Siebel Objects	50

Siebel Object Definitions	51
Object Types and Parent-Child Relationships	52
Classes in Siebel Tools	54
Siebel Repository	54
Object Layers and Hierarchy	57
Data Objects Layer	58
Business Objects Layer	62
Logical User Interface Objects Layer	68
Physical UI Layer	74
Summary of the Major Object Types	75
Operating Architecture Overview	76
Siebel Web Engine Infrastructure	78
About Standard and High Interactivity	80
JavaScript Object Architecture in High Interactivity	81
Enabling and Disabling High Interactivity for Applications	83
Enabling and Disabling High Interactivity for Views	84
High Interactivity Configuration Considerations	84
Integrating Siebel with J2EE	85
Siebel Partner Connect and Siebel Tools for Partner Connect	87

Chapter 3. Siebel Tools Fundamentals

What Is Siebel Tools?	89
Siebel Tools Features	90
Siebel Tools Application Window	90
Siebel Objects	90
Siebel Object Explorer	91
Web Layout Editors	91
Script Editors	92
Wizards	93
Target Browser Support	97
Object Repository	98

About the Object Explorer	98
Object Explorer Window	99
Showing and Hiding Objects in the Object Explorer	105
Filtering Object Types by Project	106
Object List Editor Window	107
Other Windows	108
Hiding the Windows	113
Docking the Windows	113
Image Preview	114
Drilldown	114
Viewing Object Definitions	114
Modifying, Copying, and Creating New Object Definitions	117
Object Definitions, Value Types, and Naming Conventions	117
Modifying Object Definitions	118
Creating a Copy of an Existing Object Definition	120
Creating a New Object Definition	121
Undoing New or Changed Object Definitions	123
Validating Object Definitions	123
Compiling and Testing Object Definitions	129
Compiling Projects	130
Compiling Single Objects or a Group of Objects	131
Compiling the Siebel Repository Using the Command-Line Interface	132
Testing Repository Changes	132
Setting Up Debug Options	133
Understanding the Changed Flag and Pencil Icon	135
Using Queries to List Object Definitions	137
Simple Queries	138
Compound Queries	139
Searching the Repository for Object Definitions	140
Getting Reports About Object Relationships	143

Viewing Object Relationships: Visualization Views	145
Details Visualization Views	146
Relationships Visualization View	149
Descendents Visualization View	152
Web Hierarchy Visualization Views	153
Siebel Tools Product Components	154
Siebel Object Interfaces	154
Siebel Database Extension Designer	155
Siebel Application Upgrader	155
Siebel Upgrade Inheritance	156
Siebel Object Comparison and Synchronization	156

Chapter 4. Application Configuration (Basic Concepts)

About Configuration	159
Usage and Configuration of Non-Licensed Objects	160
Configuration Goals and Objectives	160
Overview of the Web Configuration Process	161
Planning Considerations	163
Overview of the Application Development Process	164
Siebel Object Definition Sequence	166
Application Enhancement Through Scripting and Object Interfaces	171
Server-Side Scripting	172
Browser-Side Scripting	173
Generating Browser Scripts	176
Localization	178
Locale Object Types	178
Siebel Tools Language Mode	179
Check In/Out	180
Locale Management Utility	180
Controlling Visibility Using Siebel Tools	180

Visibility Establishment Process	180
Visibility Property Settings in Siebel Tools	182
Security Considerations	183
Other Ways to Customize Application Behavior	184
Personalizing Your Web Application	185
Managing Web Content with Siebel eBriefings	186
Dynamic Data Capture with Siebel eSmartScript	186
Siebel Assignment Manager	187
Siebel Business Process Designer	187
State Model	188
Siebel ePricer	188

Chapter 5. Data Objects Layer

Data Object Types	191
Tables	193
Base Tables	194
Properties of the Table Object Type	195
Data Tables	196
Extension Tables	196
Intersection Tables	204
Column Objects	215
Column Object Type	216
Data Columns	218
Extension Columns	218
System Columns	222
Indexes	225
Index Column Object Type	226
User Keys	227
EIM Interface Tables	228
EIM Object Types	229
Labeling Data Loaded in EIM As No Match Row Id Instead of NULL	238

Access Control	238
Party	240
Person-Related Data	240
Organization-Related Data	242
Party Business Components	245
S_PARTY Table	246

Chapter 6. Adding Custom Extensions to the Data Model

About Extending the Siebel Data Model	251
About Standard Database Extensibility	252
Using Standard Database Extensibility	254
Database Extension Planning and Design	256
Planning and Design Steps	256
Naming Conventions for Extension Tables and Columns	260
Accommodating Active Mobile Clients	261
DBMS Restrictions	261
Database Extension Implementation	262
Checking Out and Locking the Projects	263
Updating the Logical Schema Definition in the Local Environment	264
Creating a Custom Extension Table	264
Adding Extension Columns to Tables	266
Creating Extension Columns of Type LONG	268
Modifying Extension Tables or Columns	269
Deleting Extension Tables or Columns	270
Using Extensions with Enterprise Integration Manager	272
Adding Custom Indexes	274
Applying the Physical Schema Extensions to the Local Database	275
Displaying Extension Data	277
Displaying Base Table Extension Column Data	278
Displaying Data in One-to-One Extension Tables	279

Displaying Data from One-to-Many Extension Tables	279
Preparing the Server Database Before Applying Schema Extensions	280
Applying the Changes to the Server Database	281
Applying Schema Changes Using Siebel Tools	281
Applying Schema Changes Using the Configuration Utility	283
Applying Server Database Changes to Other Local Databases	289
Populating Extension Tables and Columns	290
Making Extension Tables Available for Population by EIM	290
Configuring Client-Side Import	291
Advanced Database Extensibility	292
Creating New Tables Using the Table Wizard	293
EIM Table Mapping Wizard	299
Dock Objects	312
Dock Object Types	312
Dock Object Tables	314
Dock Object Visibility Rules	316
Finding the Dock Object for a Business Component	318
Docking Wizard	320
Creating a New Dock Object	325
Adding a New Dock Table to an Existing Custom Dock Object	328
Verifying Dock Objects	332
Deleting and Cleansing Dock Objects	333
Consulting Siebel Expert Services	333

Chapter 7. Business Objects Layer

Major Business Object Types	338
Usage and Configuration of Non-Licensed Objects	341
Business Components	342
Base Tables of Business Components	344
Joined Tables and Extension Tables of Business Components	345

Sort Specification Property	347
Search Specification Property	348
Configuring Data-Driven Read-Only Behavior	350
Intersection Business Components	356
Virtual Business Components	358
Master-Detail Business Components	360
Fields	362
System Fields	366
Calculated Fields	367
Field Data Types	368
Sequence Fields	373
Joins	378
How a Join Is Constructed	382
Using a Predefault Value for a Join Field	385
Party Business Components and Joins: Party Extension Tables	386
Mapping Fields in Party Business Components	387
Bringing Party Data into a Non-Party Business Component	388
Bringing Party Data into Party Business Components	390
Mapping a Field to a Column in a Party Table	392
Links	393
How a Link Is Constructed	396
Using a Link in a Master-Detail View	398
Using a Link in a Multi-Value Group	399
Using a Link in a Many-to-Many Relationship	399
Using a Link When Merging Records	399
Cascade Delete Property	399
Multi-Value Links	400
How a Multi-Value Link Is Constructed	403
How an Indirect Multi-Value Link Is Constructed	408

Primary ID Field	413
How a Cascade Copy with a Multi-Value Link Is Constructed	417
About Business Objects	418
How a Business Object Is Constructed	423
Business Services	425

Chapter 8. Defining Business Objects and Business Components

About the Application Development Process	427
Defining Business Objects	428
Usage and Configuration of Non-Licensed Objects	429
Development Sequence for Defining Business Objects	430
Creating or Modifying a Business Component Definition	431
Adding Fields to a Business Component	433
Configuring Dual Currency Support	436
Creating or Modifying a Business Object Definition	439
Mapping Business Components to Business Objects	441
Using Wizards to Create Objects	443
Business Component Wizard	443
OLEDB Rowset Wizard	445
Integration Object Wizard	446

Chapter 9. Logical User Interface Objects Layer

Major User Interface Object Types	448
Applets	453
Types of Applets	454
Form Applets	462
List Applets	472
About HTML Control Types	479
About the Display Format Property	484

About the Type Property	485
About the Search Specification Property	486
Views	488
List-Form Views	489
Master-Detail Views	490
Thread Bars	493
Drilldown Behavior in a View	494
Applet Toggles	499
Screens	502
About The User Interface Navigational Paradigm	504
Applications	508
Web-Related Objects	509
Search and Find Objects	512
Toolbars and Menus	513
Toolbar and Menu-Related Object Types	515
Activating and Suppressing Menu Items and Toolbars	520
Invoke Method Targeting	521
Icon Maps	524
Specifying a Default Icon in an Icon Map	525
HTML Hierarchy Bitmap	526

Chapter 10. Logical User Interface Objects Configuration

User Interface Object Definition Sequence	529
About Defining Applets	530
About Applet Properties	534
Defining List Applets	536
Defining Form Applets	540
Editing the Web Layout of Applets	544
About Grid Layout	547

Converting Applets to a Grid-Based Layout	548
Converting One or More Applets	548
Converting Applets By Changing the Web Template	550
About Grid Layout Conversion Error Messages	552
Supported Applet Classes and Applet Web Templates	554
Editing Applets Based on Grid Layout Templates	555
Positioning Controls	557
Aligning Controls	557
Resizing Controls	558
Spacing Controls	559
Centering Controls	560
Setting Tab Order	561
Resizing the Grid Canvas	561
Setting a Default Method for an Applet	562
About Views	562
Creating Views Using the View Wizard	563
Creating Views Using the Object Explorer	564
Editing the Web Layout of Views	566
Configuring Views for Personal Layout Control	567
Providing User Access to a New View	570
Reasons a View Is Not Visible to a User	570
About Screens	572
Defining Screens	573
About Applications	576
Associating Screens with Page Tabs	577
Defining Screen Menu Items	578
About Web Pages	579
Editing the Layout of Web Page Objects	579
Associating Images With Siebel Objects	582

Defining Toolbars and Menus	584
Creating Command Objects	586
Using the Command Object Wizard	586
Using JavaScript to Extend Toolbars	590
Creating Applet Menus	591
Configuring Keyboard Accelerators	594
Adding a New Keyboard Accelerator	594
Modifying the Key Sequence for an Existing Accelerator	595
Hiding The Key Sequence in the User Interface	596
Design Considerations for Keyboard Accelerators	596
Configuring Spell Check	597
Using the Locale Management Utility	601
Finding Untranslated Text Strings	602
Finding Existing Translations	603
Exporting Strings and Other Locale-Specific Attributes	605
Importing Strings and Other Locale-Specific Attributes	605
Identifying Modified Objects	608
Using the LMU to Replace Strings	609
Running the LMU From the Command Line	610

Chapter 11. Pick Applets and Static Picklists

Pick Applets	613
Configuring the Originating Applet	620
Configuring the Pick Applet	622
Using the Pick Applet Wizard	623
Configuring the Originating Business Component	627
Configuring Pick Business Components	633
Configuring Picklists	634
Creating a Picklist Using the Pick List Wizard	636
Static Picklists	638
Configuring Originating Applets	643
Configuring Originating Business Components	643

Configuring the Pick List	644
Creating a Static Picklist Using the Pick List Wizard	645
The PickList Generic Business Component	647
Hierarchical Picklists	648
Pop-Up Visibility Rules	650
Working With Lists of Values	651
Multilingual Lists of Values	654
About the Language Independent Code	655
Enabling the Multilingual List of Values	657
Integration Considerations	673
Configuration Considerations	674
MLOV Configuration and Coding Guidelines	675
Querying and Multilingual Lists of Values	676
Configuring Siebel Business Process Designer to Use MLOV-Enabled Fields	676
Configuring Siebel Assignment Manager to Use MLOV-Enabled Fields	681
Configuring Siebel Anywhere for Use with MLOV-Enabled Fields	683
Administering the Multilingual List of Values	684

Chapter 12. Multi-Value Group and Association Applets

Multi-Value Group Applets	688
Configuring the Originating Applet	692
Configuring the Originating Business Component	692
Configuring the Multi-Value Link	693
Configuring Links	695
Configuring the Multi-Value Group Business Component	695
Using the MVG Wizard	696
Configuring the Multi-Value Group Applet	699
Using the MVG Applet Wizard	700
Association Applets	702
Association Applets Invoked from Master-Detail Views	707
Association Applets Invoked from Multi-Value Group Applets	711

Chapter 13. Special-Purpose Applets and Controls

Chart Applets	718
Axis Terminology	720
Chart Layout Options	722
Configuring Chart Applets	737
Performance Considerations	751
Using the Chart Applet Wizard	751
Tree Applets	756
Configuring Tree Applets and Explorer Views	760
Tree Applets in the Applet Web Template Layout Window	764
Recursive Trees	764
File Attachment Applets	765
Configuring Attachment Applets	768
Configuring Attachment Business Components	769
Configuring Attachment Tables	771
Pop-Up Windows	772
Configuring Pop-Up Applets Launched from Applets	773
Configuring Pop-Up Wizards	774
Configuring Pop-Up Views Launched from Applets	775
ActiveX Controls	776
Creating DLL and Class Objects That Reference an ActiveX Control	776
Adding an ActiveX Control to an Applet	778
Setting Properties in an ActiveX Control	780
ActiveX Methods and Events	782
Distributing ActiveX Controls	783
HTML Content Controls	783
Control Properties	784
Administration Views	785
Configuring Fields to Use Web Content Assets	787

Chapter 14. Physical User Interface Layer

Understanding Siebel Templates	789
Web Template Explorer	794
Understanding Siebel Tags	796
Mappings Between Controls and IDs	796
Singleton and Multi-Part Tags	797
“This” Tag	798
Iterators	798
Nesting and Siebel Tags	800
SWE Conditional Tags	800
High Interactivity Versus Standard Interactivity	800
Navigational Constructs	802
Primary Tab Bar	803
Visibility Picklist and Detail Tab View Bar	804
Subcategory Views	809

Chapter 15. Physical UI Navigation and Templates

Page Templates	815
Web Page-Layout Container Page	816
HTML Frames	817
Support for Multiple Views on a Page	822
View Templates	825
Applet Templates	828
Form Applets	829
List Applets	833
Persistently Editable List Applets	834
Displaying Totals of List Column Values	846
Multi-Value Group and Pick Applets	849
Toggle Applets	849
Tree Applets	853

Chart Applets	860
Catalog-Style List Applets and Rich Lists	862
Toolbars and Menus	865
Toolbar Template Configuration	866
Menu Template Configuration	868
Thread Bar	871

Chapter 16. Special Behavior Supported by Templates

Search and Find Configuration in SWE Templates	875
Search and Find Applet Tags	875
Results Applet Tags	877
Favorites (Predefined Queries)	879
Conditional Tags	880
SWE Conditional Tags	880
Designing Browser Group-Specific Templates	884
Conditional Mappings for Applets	887
Browser-Specific Mappings	888
Application-Specific Mappings	890
More/Less Mode-Specific Mappings	891
Image Support	891
Configuring Images as Bitmap Objects	892
Image Formats	893
Using Icons for Field Values	894
Using Images as Links in Controls	897
Image Caching File Manager	898
Hierarchical List Applets	898
About Grid Layout Templates	900
Creating Custom HTML Control Types	903
Removing HTML Frames From Web Templates	908
Modifying Page Containers	909

Modifying Headers and Footers	909
Modifying Views with Custom Headers and Footers	909
Known Issues When Running Siebel Applications Without HTML Frames	910
Template Configuration Features	910
Displaying Server Side Errors	911
Adding Graphics	913
Creating Directories for Your Graphics Files	913
Adding Sorting Capabilities to Your Application	913
Cascading Style Sheets	915

Chapter 17. Repositories

Code Pages and Unicode	917
Exporting and Importing Repository Objects	918
Exporting Individual Object Definitions	919
Exporting Object Definitions Using the Command Line Interface	921
Exporting Entire Projects	921
Importing Object Definitions	923
Importing Object Definitions Using the Command Line Interface	931
Renaming, Deleting, Backing Up, and Migrating Repositories	932
Renaming and Deleting Repositories	932
Backing Up and Restoring Repositories	934
Using repimexp.exe for Importing, Exporting, and Creating a File Dump	939
Migrating Repositories and Schemas Between Databases	942
Creating Patches	951
Creating a Patch File	952
Applying a Patch File	955
Integrating with External Source Code Control Software	957
Enabling the Interface	958
Configuring the srcctrl.bat File	959
Microsoft Visual SourceSafe Examples	964

Check In/Check Out Options (Source Control Integration)	965
Upgrading Repositories: Siebel Application Upgrader	966
Web Client Migration Wizard	968
Automatic Upgrade of Copied Objects	968
Upgrade Inheritance Scenario	970
Recommended Guidelines for Copying Objects	970
How Enhancements Are Applied During an Upgrade	972
Repository Location of the Upgrade Ancestor	972
Configuration Steps for Upgrade Inheritance	973
Object Comparison and Synchronization	975
Viewing the Object Comparison Dialog Box	977
Differences Between Checked-Out Projects	981
Entering the Comparison	983

Chapter 18. Application Development Projects

What Are Siebel Projects?	987
Getting Information About Repositories and Projects	989
Selecting the Current Repository	989
Getting Information About the Current Repository	990
Viewing Object Definitions by Project	992
Getting Projects	993
Checking Out Projects	995
Creating New Projects	998
Renaming Projects	999
Assigning Object Definitions to Projects	999
Moving Object Definitions Between Projects	1000
Checking In Projects	1001
Check In Dialog Box	1003
Check-In Guidelines	1004

Check-In/Check-Out Options (Data Sources)	1005
Determining Project Differences at Check-In Time	1008
Undoing Check Out	1009
Locking Projects Directly	1010
Project Structure Considerations	1012

Appendix A. Configuring the Customer Dashboard

Understanding the Customer Dashboard	1015
How the Customer Dashboard is Populated With Data	1015
Architecture	1016
Predefined Behavior	1017
Enabling the Customer Dashboard	1017
Configuring the Customer Dashboard	1018
Adding a Business Component to the Dashboard Business Object	1019
Adding a Business Component Lists to the Dashboard Business Service	1019
Mapping Business Component Fields to the Customer Dashboard	1020
Creating Field Labels	1022
Formatting Phone # Fields	1023
Configuring the GoTo View Drop-Down List	1024
Configuring Labels for GoTo Views	1025
Modifying the Look and Feel of the Customer Dashboard	1026
Changing the Background Color and Border	1026
Changing the Size and Location	1027
Configuring Communication Events	1027
Configuring SmartScripts	1029
Activating the SmartScript Player	1029
Mapping SmartScript Variables to Customer Dashboard Fields	1030
Configuring SmartScripts to Save Answers	1030
Using Siebel VB Script and eScript	1032
Customer Dashboard Commands	1032

Siebel eScript Example	1034
Siebel VB Example	1035
About Dual Personalization	1036

Index

Introduction

This guide provides a reference book for Siebel Tools providing information on Siebel Tools architecture, object layers, configuration, templates, tags, and repositories.

This book will be useful primarily to people whose titles or job descriptions match one of the following:

Database Administrators	Persons who administer the database system, including data loading, system monitoring, backup and recovery, space allocation and sizing, and user account management.
Siebel Application Administrators	Persons responsible for planning, setting up, and maintaining Siebel applications.
Siebel Application Developers	Persons who plan, implement, and configure Siebel applications, possibly adding new functionality.
Siebel System Administrators	Persons responsible for the whole system, including installing, maintaining, and upgrading Siebel applications.

Product Modules and Options

This guide contains descriptions of modules that are optional and for which you may not have purchased a license. Siebel's Sample Database also includes data related to these optional modules. As a result, your software implementation may differ from descriptions in this Bookshelf. To find out more about the modules your organization has purchased, see your corporate purchasing agent or your Siebel sales representative.

To do the tasks described in this book, you need to have a thorough understanding of:

- The Siebel application environment and data model
- The Microsoft Windows operating environment
- Application development concepts and processes

- Relational database concepts
- Object-oriented application design

Configuring Siebel applications using Siebel Tools includes many or all of the following tasks; you should have prior experience in doing similar tasks:

- Installing Siebel applications and Siebel Tools
- Setting up the Siebel application development environment (for example, workstations and servers)
- Installing and setting up the relational DBMS where the native data is stored
- Importing existing native data into your Siebel application
- Modifying or creating new Siebel object definitions
- Writing Siebel VB or Siebel eScript application code

Product Modules and Options

This Siebel Bookshelf contains descriptions of modules that are optional and for which you may not have purchased a license. Siebel's Sample Database also includes data related to these optional modules. As a result, your software implementation may differ from descriptions in this Bookshelf. To find out more about the modules your organization has purchased, see your corporate purchasing agent or your Siebel sales representative.

How This Guide Is Organized

This guide provides information necessary to implement, configure, and monitor Siebel eBusiness Applications. It contains guidance information, basic and advanced concepts, and reference information. It presents installation and basic concepts first. Then it covers configuration at the various layers of the Siebel object model—data objects layer, business objects layer, user interface layer, and physical user interface layer. Finally, it covers topics such as managing repositories, managing projects, and performance considerations.

To access SupportWeb, you will need to provide the user name and password you received from Siebel Support Services (support@siebel.com).

Application development reference information is also available in the following books, available on *Siebel Bookshelf* and in *Siebel Tools Online Help*:

- *Siebel Developer's Reference*
- *Object Types Reference*
- *Siebel Object Interfaces Reference*
- *Siebel eScript Language Reference*
- *Siebel VB Language Reference*

Revision History

Siebel Tools Reference

Version 7.5.3 Rev A

Table 1. Changes Made in Version 7.5.3 Rev A

Topic	Revision
“Siebel Tools ODBC Data Sources” on page 40	Made correction to the caution note at the end of the table.
“About Standard and High Interactivity” on page 80	Revised section for clarity.
“High Interactivity Configuration Considerations” on page 84	Added section.
“Extension Tables” on page 196	Added paragraphs three and four for clarity.
“Siebel Architecture (Basic Concepts)”	Removed the sections “View Layout Caching” and “Persistent View Layout Caching.” These sections are now included in <i>Performance Tuning Guide</i> .
“Configuring Client-Side Import” on page 291	Revised the first paragraph to include more information about what objects support client-side import.
“Intersection Business Components” on page 356	Added information about displaying data from denormalized columns in paragraph six.
“Using the Check No Match Property with a Primary Join” on page 416	Added paragraph nine clarifying the use of the property <i>Use Primary Join</i> .
“Configuring Dual Currency Support” on page 436	Corrected errors in step four of the second procedure in the topic.
“Applet Toggles” on page 499	Added the fourth bullet point in the second paragraph.
“Logical User Interface Objects Configuration”	Deleted the topic “Using the Web Layout Wizard” because it is obsolete.
“Editing the Web Layout of Applets” on page 544	In step six of the procedure, revised the third row under the Action column.

Table 1. Changes Made in Version 7.5.3 Rev A

Topic	Revision
“Editing Applets Based on Grid Layout Templates” on page 555	Added the second bullet after the fourth paragraph.
“Hierarchical Picklists” on page 648	After step two, added substep b.
“MLOV Configuration and Coding Guidelines” on page 675	Added first bullet about LookupName and LookupValue function.
“File Attachment Applets” on page 765	Corrected text in the third paragraph.
“Exporting Object Definitions Using the Command Line Interface” on page 921	Corrected the command name in the first paragraph.

Additional Changes

- Removed references to specific browsers.
- Deleted the chapter on Performance. This content now appears in *Performance Tuning Guide*.

Version 7.5.3**Table 2. Changes Made in Version 7.5.3**

Topic	Revision
“Verify Read/Write Access to Tools Directories” on page 40	Added section.
“Setting Up Developers as Mobile Users” on page 45	Revised for 7.5.3: Added content after step 5.
Persistent View Layout Caching	New for 7.5.3: Added section to support new functionality. Note that this section was moved for 7.5.3, Rev A. See Table 1 .
“Showing and Hiding Objects in the Object Explorer” on page 105	Revised for 7.5.3: Revised content to support new functionality.
“Compiling and Testing Object Definitions” on page 129	Revised for 7.5.3: Revised content to support new functionality.
“Setting Up Debug Options” on page 133	Revised content.

Table 2. Changes Made in Version 7.5.3

Topic	Revision
“Siebel Tools Language Mode” on page 179	Added note after the first paragraph.
“Creating a Custom Extension Table” on page 264	Revised content in paragraphs preceding the procedure.
“Adding Extension Columns to Tables” on page 266	Added note.
“Creating Extension Columns of Type LONG” on page 268	Added new section.
“Adding Custom Indexes” on page 274	Revised content.
“EIM Table Mapping Wizard” on page 299	Added note and revised content after step 6.
“Virtual Business Components” on page 358	Added bullet point about support for dynamic toggle applets.
“Configuring Dual Currency Support” on page 436	Corrected procedure and added note.
“Form Applet Control Properties” on page 535	Removed item for Tab Stop property.
“Defining Form Applets” on page 540	Revised for 7.5.3: Revised content to support functionality.
“Applet Toggles” on page 499	Revised content and added a configuration scenario.
“Dynamic Drilldown Behavior” on page 496	Revised content.
“Editing the Web Layout of Applets” on page 544	Revised for 7.5.3: Revised content to support functionality.
“About Grid Layout” on page 547	New for 7.5.3: Added content to support new functionality.
“Converting Applets to a Grid-Based Layout” on page 548	New for 7.5.3: Added content to support new functionality.
“Editing Applets Based on Grid Layout Templates” on page 555	New for 7.5.3: Added content to support new functionality.

Table 2. Changes Made in Version 7.5.3

Topic	Revision
“Configuring Keyboard Accelerators” on page 594	Revised section.
“Configuring Chart Applets” on page 737	Added note after Series Field item.
“Pop-Up Windows” on page 772	Revised content.
“Page Templates” on page 815	Revised definitions of Acknowledgement Web Page and Acknowledgement Web View.
“More/Less Mode-Specific Mappings” on page 891	Added note.
“About Grid Layout Templates” on page 900	New for 7.5.3: Added content to support new functionality.
“Getting Projects” on page 993	Revised for 7.5.3: Revised content to support new functionality.
“Checking Out Projects” on page 995	Revised content.

Version 7.5, Rev. A

Table 3. Changes Made in Version 7.5, Rev. A

Topic	Revision
“To implement database extensions” on page 263	Corrected step 6.
“Creating a Custom Extension Table” on page 264	Revised section.
“Adding Extension Columns to Tables” on page 266	Added a note after the procedure and revised caution note.
“To map a new table to an EIM interface table using the EIM Table Mapping Wizard” on page 300	Added explanatory text after step 6.
“Association Applets” on page 702	Corrected the name of the event referenced in the last note in the section.

Table 3. Changes Made in Version 7.5, Rev. A

Topic	Revision
“Multi-Record Select List Applets” on page 845	Corrected first and second paragraphs.
“Removing HTML Frames From Web Templates” on page 908	Added this topic.
“Exporting and Importing Repository Objects” on page 918	Added the fourth bullet.
“Migrating Repositories and Schemas Between Databases” on page 942	Added the second note.
“Compiling the Siebel Repository Using the Command-Line Interface” on page 132	Corrected the first paragraph.

This chapter explains how to install Siebel Tools and set up the application development environment.

- [Preinstallation Tasks on page 31](#)
- [Installing Siebel Tools on page 32](#)
- [Postinstallation Tasks on page 37](#)

Preinstallation Tasks

Complete the following tasks before running the Siebel Tools installation program.

- [“Verifying Siebel Tools Prerequisites”](#)
- [“Installing a Database Server or Sample Database”](#)

Verifying Siebel Tools Prerequisites

Before you install Siebel Tools, make sure that:

- Your client platform meets the requirements defined in *Siebel System Requirements and Supported Platforms* and the Release Notes for the version of the software you are using.

These documents can be found on Siebel SupportWeb at <http://ebusiness.siebel.com/supportweb/>

- You have all the third-party software installed, including the database connectivity software for your chosen RDBMS, required for your implementation.

Installing a Database Server or Sample Database

As a prerequisite to using Siebel Tools, you must have installed a Siebel Database Server or the Sample SQL Anywhere database included with Siebel eBusiness Applications. These databases are used to store the Siebel Tools project repositories.

NOTE: The sort order for the server database should be set to Binary.

For more information about installing and configuring a Siebel Database Server see *Siebel Server Administration Guide* and the *Siebel Server Installation Guide* guide for the operating system you are using.

Installing Siebel Tools

You install Siebel Tools from the *Siebel eBusiness Applications, Siebel Tools* CD-ROM.

To install Siebel Tools on a workstation

- 1** Insert the *Siebel eBusiness Applications, Siebel Tools* CD-ROM in the CD-ROM drive of your computer.
- 2** Navigate to the `siebel_tools_lep_2` directory, and then double-click `install.exe`.
The Siebel Tools installation program starts.
- 3** For setup language, choose English.
- 4** In the Welcome dialog box, click Next.
- 5** In the Setup Type dialog box, select the type of Siebel Tools installation to install on this computer:
 - **Typical Setup.** Install all Siebel Tools components. This option is recommended for most users. This option does not install the report source code which is required for creating custom reports.
 - **Compact Setup.** Install all modules except the help files and report source code.

- **Custom Setup.** Lets you customize your installation by choosing among different components. Each component is listed with the amount of disk space it requires. Click the Disk Space button to see how much disk space is available on the hard drive and network drives that are accessible from this PC. This option is recommended for experienced administrators only. If you wish to create custom reports you must perform a custom setup and check the report source code component.

NOTE: A warning appears if there is insufficient disk space to install Siebel Tools on the destination host machine. In this case, you must free some disk space before continuing with the installation process.

- a Select a destination directory.

The default directory for Siebel Tools is C:\sea7xx\tools.

The Installer does not permit installing Siebel Tools into a directory path containing more than 18 characters (for example, M:\SIEBEL\10910\TOOLS).

CAUTION: Do not install Siebel Tools in the same directory as the Siebel client. Doing so will cause memory conflicts and program crashes.

- b Click Next in the Setup Type dialog box to accept the default directory.
- c Click Browse to specify a different destination directory.

The directory name cannot be more than 19 characters long, but long filenames, including spaces, are supported. If you specify a directory other than C:\sea7xx\tools, make the appropriate substitutions as you read this guide.

- 6 Choose the installation language.

NOTE: Tools must be installed with the English language pack. If you need to customize non-English reports, you can install other language packs as well. The files specific to the languages chosen in this step are copied to your workstation.

- 7** Select the database client and server version.
 - a** Specify the name of your server database in the Server Database dialog box.
 - b** Click Next.

NOTE: The installation program checks that the prerequisite database software is installed on the machine. If it is not, the installation will not proceed.

- 8** Confirm the Siebel File System and Siebel Remote Server paths, to which this client will connect, or specify different ones.

These pathnames or values should be listed in the Deployment Planning Worksheet, which is part of *Siebel Server Administration Guide*. You must use the network name (machine name) of the server where the Siebel Server is installed. Use either the UNC name of the Siebel File System directory or a drive letter mapped to it.

- 9** Specify the database identification.

- For DB2, complete the following:

Database Alias. Enter the database alias for your Siebel Server Database in the Deployment Planning Worksheet.

Tableowner. Enter the name of the database account that owns the Siebel tables in the Deployment Planning Worksheet.

- For Microsoft SQL Server, complete the following:

Database Alias. Enter the server name for your Siebel Server Database in the Deployment Planning Worksheet.

Database Owner Name. Enter the name of the database owner account in the Deployment Planning Worksheet.

NOTE: Microsoft SQL Server is case-sensitive; all information must be entered exactly as it exists on the SQL Server database.

- For Oracle, complete the following:

Database Alias. Enter the SQL*Net connect string for your Siebel Server Database. This is recorded on Page 5, Section 11 of the Deployment Planning Worksheet.

Table Owner. Enter the name of the database account that owns the Siebel tables. This is also recorded on Page 5, Section 11 of the Deployment Planning Worksheet.

NOTE: Make sure the database is configured to handle binary sort order. For more information, see the *Siebel Server Installation Guide* for the operating system you are using.

- 10** Select the program folder. The Siebel Tools icons are installed in this directory. The default is Siebel Tools 7.xx If you do not want to accept the default:

- Select a folder different from the default from the list, or
- Type in a new program folder name.

The Event Log dialog box appears after the installer has completed copying files. It describes the steps the installer completes during your Siebel Tools installation.

Language Pack Installation

The Siebel Installer automatically begins the Language Pack installation process.

Follow the prompts for language and database choices.

ODBC Installation

NOTE: If your PC has the exact required versions of the ODBC Text and Access drivers (and the Microsoft SQL Server, if you are deploying on that database), the ODBC Pack dialog box does not appear.

- If you are not required to complete the ODBC Pack dialog box, you see a dialog box that shows the progress of ODBC component registration.
- You may be requested to complete the Required Software Component - ODBC Pack dialog box.

- If you have newer versions of the ODBC drivers installed, this dialog box lets you choose whether to install the older versions of the ODBC drivers included with the Siebel application software.
- If you have older versions of the ODBC drivers installed, you must install the versions provided with Siebel eBusiness Applications before continuing with the installation.

To install the Microsoft Data Access Components

- 1** Click Next to launch the Microsoft Data Access Components (MDAC) installer, which installs the ODBC drivers.

NOTE: Microsoft licensing requires that at least one 32-bit Microsoft product, such as Microsoft Word 7 (the version of Word that comes with Office 97), be installed on the PC before Data Access Components can be installed.

- 2** Click Yes to accept the license agreement.
- 3** Click Continue to proceed with the installation.
- 4** Click Complete to install the required ODBC drivers.
- 5** If you selected Microsoft SQL Server in the Server Database dialog box, the Local Database dialog box will appear.
 - a** Specify the local database used by your organization for Siebel Remote mobile users.
 - b** Click Next.

NOTE: If your server database is not Microsoft SQL Server, you will not see this dialog box. Proceed to the next step.

- 6** Click OK to exit the installer and return to the Siebel Tools installation program.

NOTE: Do *not* select Restart Windows if prompted to do so by the MDAC installer. Instead, select Exit Setup. Siebel Systems recommends that you complete the installation before rebooting your computer.

The Siebel Tools installation is now complete.

Postinstallation Tasks

Do the following tasks (described in the following sections) after running the Siebel Tools installation program:

- “Verifying Successful Installation”
- “Verifying the Siebel Tools Directory Structure” on page 38
- “Siebel Tools ODBC Data Sources” on page 40
- “Running Multiple Local Databases” on page 41
- “Repository Naming Conventions” on page 43
- “Setting Up the Development Environment” on page 44

Verifying Successful Installation

Verify that installation was successful by connecting to the Siebel Database Server and entering your license key for Siebel Tools.

To verify that the installation completed successfully

- 1** Start Siebel Tools and log onto the Siebel Database Server.

The first time you log on, the system prompts you to enter a license key number if you have not done so already.

- 2** Enter your license key number in the dialog box that appears, and click OK.

You can find the license key that has been assigned to your site in your License Key letter provided with the CD-ROM case. For more detail on license keys, refer to *Applications Administration Guide*.

Verifying the Siebel Tools Directory Structure

The Siebel Tools installation program creates the following directories.

TOOLS. The Siebel Tools root directory.

NOTE: The root directory name may differ, based on installation choices made.

actuate	Actuate-related files for generating and running reports.	
	afc	Actuate Foundation Class files.
	bin	Actuate binary files.
	cache	Actuate cache files.
bin	All binary files (.exe, .dll, .cfg, .dsn, .enu, .bat), configuration files, and user preference files.	
	enu	Language-specific dll files.
	dll	Siebel Tools program library files.
	bin	Siebel Tools binary files.
	exe	Siebel Tools executable files.
classes	Java code files.	
	examples	
	examples\src	
	examples\src\com	
	examples\src\com\siebel	
	examples\src\com\extra	
	examples\src\com\integration	
	examples\src\com\integration\mq	Examples of Java code files.
	examples\src\com\integration\servlet	Examples of Java code files.
help	Siebel Tools help files.	
	enu	American English language help files.

local	The local, extracted database.	
	files	Local file attachments.
	inbox	Not used for Siebel Tools.
	outbox	Not used for Siebel Tools.
locale	Language-specific files.	
log	Log files from Siebel Tools operations.	
msgtempl	Message files.	
objects	Contains <code>siebel.srf</code> , the compiled definition file used by Siebel Tools. This is also the default location for <code>.srf</code> files created using the Siebel Tools object compiler.	
public	HTML and graphics files for uploading, and cascading style sheet files.	
reports	All report files.	
	enu	American-English version of reports.
rptsrc	Actuate source files for all reports.	
	enu	American-English version.
	enu\lib	Report object library (.rol) files.
	enu\standard	Report object design (.rod) files.
	lib	Report object library (.rol) files.
	standard	Report object design (.rod) files.
sample	Where the sample database (<code>sse_samp.dbf</code>) is installed.	
	files	Where the sample file attachments are installed.
	inbox	Not used for Siebel Tools.
	outbox	Not used for Siebel Tools.
scripts	Location of Java Scripts.	
search	Location of saved searches, in target-language and in language-independent formats.	
	enu	
	enu\data	

enu\data\html
enu\data\other
enu\index
enu\scripts
language-independent

- sqltemp** SQL statement fragments used by certain Siebel Tools operations.
- temp** Temporary working area.
- upgrade** Not used for Siebel Tools.
- webtempl** Location of Siebel Web Template (.swt) files.

Verify Read/Write Access to Tools Directories

After installing Siebel Tools, verify that the Tools user has sufficient permissions to read and write data to the Tools directory. You can review the permission settings by selecting the Siebel Tools root directory, right clicking, and then choosing Properties. Permissions are listed in the Securities tab.

Siebel Tools ODBC Data Sources

To create system data sources, you must modify the SystemDSN parameter in the siebel.ini file before running the Siebel Tools installation. For information about how to modify the SystemDSN parameter in the siebel.ini file, see *Siebel Web Client Administration Guide*.

The Siebel Tools Installer creates the ODBC data sources described in [Table 4](#). By default, these are created as user data sources, which are visible only to the user account under which Siebel Tools is installed.

Table 4. Siebel Tools ODBC Data Sources

Data Source	Use
SSD Local Db C:/sea7xx	Connects to the local SQL Anywhere database.
SSD DB2Udb C:/sea7xx	Connects to the DB2 database.
SSD MSQL C:/sea7xx	Connects to the Microsoft SQL Server database.
SSD Oracle C:/sea7xx	Connects to the Oracle database.

CAUTION: A name for the local database that is too long (for example, SSD LOCAL DB M: /SIEBEL/10910/TOOLS) results in a data source that does not appear in the Control Panel ODBC Administration applet. It also causes the failure of such Siebel Tools operations as checkout. The total length of the data source name must be not more than 32 characters.

Running Multiple Local Databases

You may want to have multiple local databases open at the same time—for example, to run Siebel Tools and the Siebel client application simultaneously. If you want to operate in this manner, you must change the local database configuration of one of the applications.

SQL Anywhere also requires that all database files opened simultaneously have unique filenames. Because all Siebel client applications use the same default name for their local database files, you must rename the database file of one application and move it to the database directory of the other.

To rename and move the local database for your Siebel Tools installation

- 1 Exit Siebel Tools if it is running.
- 2 Edit the .cfg file to change the name of your local database file:

- a** Open the tools.cfg file (in the \bin subdirectory of your Siebel Tools installation) with a text editor.
- b** Replace the file parameter for `ConnectionString`, under the [Local] section, with the new name and location of the Siebel Tools local database file.

For example, change:

```
ConnectionString = c:\sea7xx\tools\local\sse_data.dbf
```

to:

```
ConnectionString = c:\sea7xx\tools\local\sse_tools.dbf
```

- 3** Rename the Siebel Tools local database file and move it to the directory specified in [Step 2](#).

In this example, the `c:\sea7xx\tools\local\sse_data.dbf` file would be renamed `sse_tools.dbf` and moved to the `c:\sea7xx\tools\local` directory.

- 4** Modify the ODBC data source that points to the Siebel Tools local database as follows:

- a** From a DOS prompt, navigate to the \bin subdirectory of your Siebel Tools installation.

- b** Type `ODBCAD32` to start the ODBC administrator.

- c** Select the data source used by Siebel Tools.

This defaults to SSD Local Db `c:\sea7xx`

Where:

`c:\sea7xx` = the directory into which Siebel Tools was installed.

- d** Click the Configure button.
- e** Change the value in the Database File window, under the Database Startup section, to the new location and name of the Siebel Tools local database. In this example, change

```
c:\sea7xx\tools\local\sse_tools.dbf
```

to:

```
c:\sea7xx\tools\local\sse_tools.dbf
```

- f Close the ODBC32 applet.

Repository Naming Conventions

Use a consistent naming convention for your repositories in all environments. The Siebel eBusiness Applications client and Siebel Server programs point to a specific repository by name. The procedures for upgrading to new versions of Siebel eBusiness Applications are also dependent on repository names.

A consistent naming convention assures successful configuration and testing, and minimizes the work required when migrating new repositories or performing upgrades. Follow these guidelines in determining your repository naming conventions:

- **Production environment.** Decide on a repository name to be used in your production environment. By default, the name is Siebel Repository and should be changed only if you have a compelling reason, because much of the Siebel documentation and the default configuration of Siebel eBusiness Applications assume the use of this name.
- **Test environment.** Choose the same name for the active repository in your test environment and 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, and eliminates the need to change your Siebel client or server configurations when you do so.
- **Development environment.** Use descriptive names for the other repositories in your development environment. Your development environment may contain a number of repositories in addition to the current repository that is being configured, including the initial repository loaded with Siebel, other repository versions used in Siebel upgrades, and repositories from previous versions of your custom configuration.

Give these repositories unique and fully descriptive names. For example, you might use Siebel 7.0 Original for the initial repository that was included in the standard Siebel eBusiness Applications version 7.0.

Setting Up the Development Environment

This section:

- Explains how to set up and work in the Siebel Tools development environment
- Describes how to establish the development environment
- Lists the names of important directories
- Explains how to set up developers as mobile users

For an example of setting up the developer environment, see *Developing and Deploying Siebel eBusiness Applications*.

Creating the Development Environment

As a developer, you need to know which repository is being used for the test environment, for the system test environment, and for the production environment. Operating effectively with multiple environments requires the ability to work with local databases, and a familiarity with checking projects into and out of repositories.

First, create a complete development environment that includes both a Siebel Database Server and a Siebel Server. These can reside on the same physical machine. This environment should be completely separate from your production environment—no development work should be performed in the production environment.

Create a separate test environment into which your configuration can be migrated for system testing prior to installation in the production environment. As with the development environment, the test environment should include both a Siebel Database Server and a Siebel Server.

The *development database* will store the working copies of all repositories being configured by all developers. Configuration work should take place only on the development database. After you have finished configuring a repository, you will use the Siebel Repository Migration Utility to transfer that repository to the test (and, later, production) environment. For information about this utility, see [“Backing Up and Restoring Repositories” on page 934](#).

Setting Up Developers as Mobile Users

Because all developers need Siebel Tools and a local repository, they should install the Siebel Mobile Client on their computers.

For more information on setting up local databases, see *Siebel Remote and Replication Manager Administration Guide*.

To set up developers as mobile users

- 1 Install Siebel Tools on all developers' PCs, in a directory separate from the standard Siebel applications.

For example, if you have installed your Siebel applications in C:\siebel, install the development tools in C:\sea7xx\tools. This makes sure that the development and run-time environments are distinct. You may be using Siebel Remote in both environments, so you need to make sure that the two installations do not conflict.

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

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

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

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

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

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

4 Grant each developer a position and a responsibility.

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

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

5 On the Siebel Server, generate a database template.

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

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

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

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

7 Initialize the Developer's Mobile Client Database.

Begin by double-clicking the Siebel Tools icon and connect to the local database.

8 Enter the Siebel developer logon created in [Step 3 on page 45](#) and an appropriate password.

The initialization program creates the `sse_data.dbf` local database in the `\local` directory of your Siebel Tools installation, for example `c:\sea7xx\tools`.

- 9** Do an initial get of all projects on each local database, as described in [“Getting Projects” on page 993](#).
- 10** Check Out selected projects you want to work on, as described in [“Checking Out Projects” on page 995](#).

This chapter describes the Siebel application architecture, which is a layered structure containing the physical user interface, logical user interface objects, business objects, and data objects, as well as a third-party relational database management system.

This chapter also introduces the Siebel operating architecture, which includes the Object Manager, the Web Engine, and the Data Manager. It also outlines key architectural considerations in setting up your application development project and describes the application architecture and object definition terminology that define how Siebel applications are configured.

About Siebel Tools

Siebel Tools allows you to customize Siebel applications by modifying and creating object definitions. A standard Siebel application provides a core set of object definitions, which you can use as a basis for your own tailored application.

NOTE: The terms *object* and *object definition* in the Siebel application context do not mean the same thing as the terms “object,” “object class,” or “object instance” as used in a programming language like C + +.

Object definitions are grouped into four layers with different subject matter and purposes; for example:

- Physical User Interface Layer: Templates and tags that render UI
- Logical User Interface Objects Layer: Presentation of data (UI)
- Business Objects Layer: Business entities
- Data Objects Layer: Database details (data)

Object types in a given layer depend on definitions in the next lower layer, and are insulated from other layers in the structure. This means, for example, that you can make changes to a Siebel application without changing the underlying database structure. Similarly, you can extend the Siebel database schema without impacting the Siebel application.

Application Architecture Overview

The application architecture is discussed in the following sections.

- [“Siebel Objects” on page 50](#)
- [“Siebel Object Definitions” on page 51](#)
- [“Object Types and Parent-Child Relationships” on page 52](#)
- [“Classes in Siebel Tools” on page 54](#)
- [“Siebel Repository” on page 54](#)
- [“Object Layers and Hierarchy” on page 57](#)
- [“Data Objects Layer” on page 58](#)
- [“Business Objects Layer” on page 62](#)
- [“Logical User Interface Objects Layer” on page 68](#)
- [“Physical UI Layer” on page 74](#)
- [“Summary of the Major Object Types” on page 75](#)

Siebel Objects

Siebel applications are built on object-oriented principles.

A Siebel object definition is a data construct in the repository file that defines an element of the:

- User interface
- Business entities

- Database organization

An object definition consists of a set of properties with assigned values. For example, a view might have the following properties listed in the following table:

Property	Value
Name	Account List View
Title	My Accounts
Inactive	FALSE

Siebel Object Definitions

An *object definition* in the Siebel Tools environment implements one piece of the software—a user interface, an abstract data representation, or a direct database representation or construct. For example, a database column, a pop-up window for record selection, and a join relationship between database tables are implemented as object definitions.

An object definition consists of *properties*, which are characteristics of the software construct that the object definition implements. For example, the properties of a database column include its name, data type, length, and so on. Similarly, the properties of a pop-up applet include the caption that appears in the title bar, and so on.

NOTE: Object definitions in Siebel Tools are not equivalent to objects in object-oriented programming, although Siebel object definitions are implemented using true C++ classes and objects. Terms such as object, property, or class refer to the Siebel application metadata and not to the corresponding terms in object-oriented programming.

In Siebel Tools, object definitions are viewed, edited, added, and deleted in the Object List Editor window. Each row in the Object List Editor displays an object definition; each column displays a property. The intersection of a row and column in the grid is a particular property setting for a particular object definition.

A property setting for an object definition is changed in the Object List Editor by clicking the corresponding cell. A new value can then be manually entered or, in many cases, selected from a picklist. An object definition can be added or removed in the Object List Editor by selecting the desired row, and clicking Add New Record or Delete Record in the Edit menu.

NOTE: Some objects contain Unicode information, others contain ASCII only. Objects that contain Unicode information store data in tables with a suffix of `_INTL` in the table name or in the `S_MSG` table. Objects that do not store data in these tables contain ASCII only.

Object Types and Parent-Child Relationships

An object type is a named structure from which object definitions of that type can be created. For example, the Account, Opportunity and Contact business components are implemented as object definitions of the Business Component object type. Conceptually, object types are the templates from which object definitions are created. An object type can be thought of as the cookie cutter that is used to make many cookies of a particular shape.

An object type has a predefined set of properties. Object definitions created from it have values for each of these properties (the values are either default or user-specified). For descriptions of all properties for all object types (other than Siebel-use only object types), see *Object Types Reference*.

Object types are displayed in the Object Explorer window, which operates in parallel with the Object List Editor window. Selecting a different object type in the Object Explorer causes an Object List Editor window to display object definitions for that object type. The title bar of the Object List Editor window identifies the kind (object type) of object definitions it contains. Every object definition has exactly one object type. When a new object definition is created in the Object List Editor, the object type of the active Object List Editor window determines the new object definition's object type.

You can change the property values in an object definition, but you cannot change the set of properties to which the values are assigned. The set of properties is fixed for each object type.

There is a predefined set of object types in Siebel Tools that have specific purposes. For example, two object types are Applet and Business Component. An applet object definition defines a user interface unit such as a data entry form or editable list of records. A business component object definition defines a data record structure from one or more database tables.

Object types have hierarchical relationships called *parent-child relationships*. These can be seen when you expand an object type that has children in the Object Explorer (and when the Object Explorer is in Types view). The Object Explorer in Types view uses the same visual metaphor for displaying hierarchical relationships as the Windows Explorer in Windows 2000 or NT. An object type (folder) beneath and slightly to the right of another is the child object type of the one it is below, the latter of which is the child's parent object type.

An object type can have multiple child object types. For example, if you expand Applet in the Object Explorer, you see various child object types, including Applet Method Menu Item, Applet Browser Script, Applet Server Script, Applet Toggle, and so on. Object definitions, like object types, have parent-child relationships. These relationships are based on their object types. That is, the object type of the parent object definition determines the object types of the child object definitions. Parent-child relationships between object definitions are displayed in Siebel Tools with two Object List Editor windows open simultaneously.

Figure 1 on page 53 shows the Siebel Tools window displaying business component object definitions in the upper list applet, and field object definitions in the lower list applet for the currently selected business component.

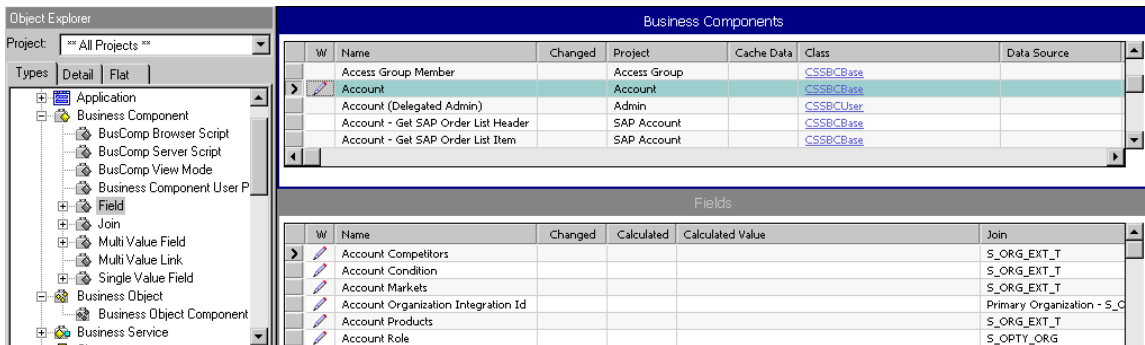


Figure 1. Object Explorer and Two Object List Editor Windows

Parent-child relationships between object definitions imply containership. That is, the child object definition is in, or belongs to the parent object definition. For example, columns are in a table, fields are in a business component, joins are also in a business component, and controls are in an applet. There is no inheritance among object definitions, and the set of properties of an object definition is unrelated to the set of properties of a child object definition.

Classes in Siebel Tools

In Siebel Tools, *Class* is a property of certain object types, such as applet and business component. Most object types do not have a Class property. The selection of a Class property value, when available, assigns a set of behaviors to the object definition, and distinguishes it from other categories of object definitions of the given object type. For example, a value of *CSSFrameList* in the Class property in an applet object definition makes it a list applet. The Class property accomplishes this by assigning a DLL to the object definition (indirectly by way of a class object definition).

Siebel Repository

The Siebel repository contains tables in the Siebel database where Siebel object definitions are stored. You view the contents of the Siebel repository through the Siebel Tools windows that appear when you initialize Siebel Tools.

The set of object definitions and server scripts defining a Siebel application (such as Siebel Service or Siebel Sales) or set of applications is compiled into a file called a Siebel repository file, or *.srf* file.

NOTE: Browser scripts are compiled into the browser script compilation folder, which can be specified in Siebel Tools on the Scripting tab under View > Options. For more information on the browser script compilation folder, see [“Browser-Side Scripting” on page 173](#).

The .srf file, when opened by the object manager on behalf of a Siebel Web application, provides the system with all of the information it needs to define interactions with the enterprise data and software users. In its uncompiled form, the set of object definitions is called a repository, and is stored in the database that Tools connects to. In its compiled form, it occupies a single compressed read-only file with the .srf file extension.

The object manager reads the information stored in the repository file on demand as the different parts of the Web application are executed. For example, when the user switches to a specific view, all components that form part of that view will read information from the repository file that determines which data will be presented and in what format. Some components are visual and the configuration information will determine, for example, what applets to show in the view. In the case of non-visual components, the configuration information may determine what data fields this component provides.

The application database contains the actual data for the system, and it also contains some administration information such as the list indicating which views a specific type of user can access. This database can run locally or remotely. The contents of the repository tables are compiled into a binary file that provides fast read-only access to the repository metadata during run-time.

[Figure 2 on page 56](#) shows the definition of the Contact repository object as viewed in Siebel Tools.

The Siebel database contains both metadata (repository objects) and user data. The database can be populated either through the Siebel UI or a variety of interfaces.

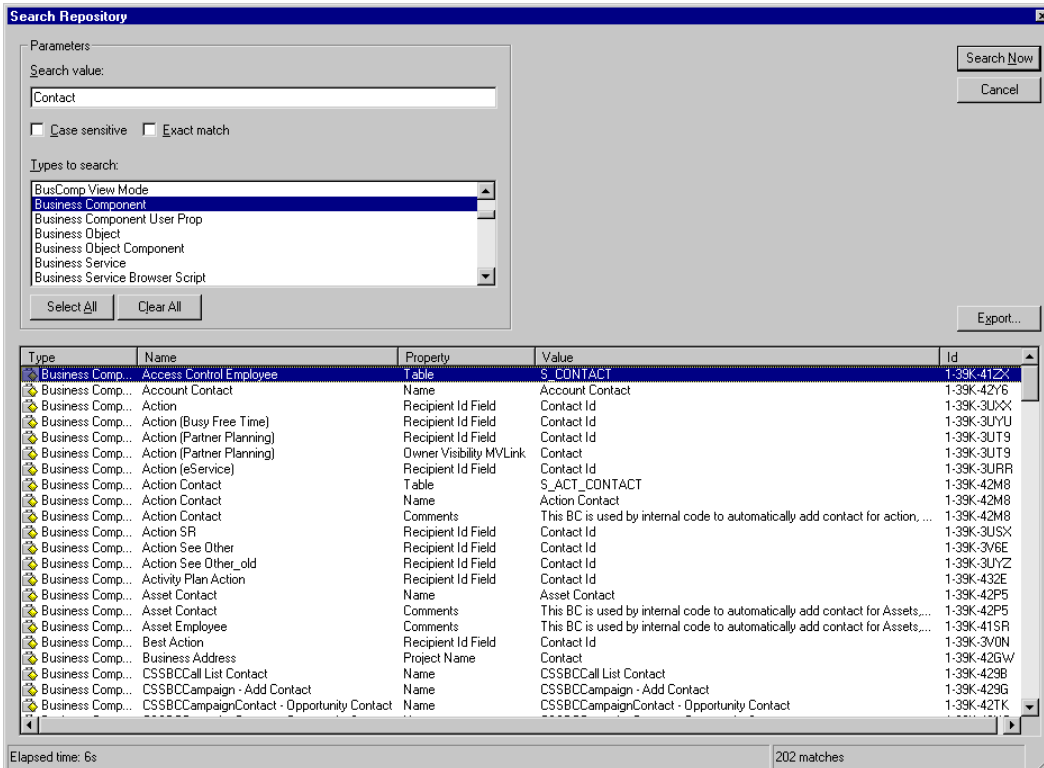


Figure 2. Contact Repository Objects as Viewed in Siebel Tools

Virtual business components allow data from external RDBMSs and applications to be displayed in the Siebel user interface.

Object Layers and Hierarchy

The object definitions in Siebel applications fall into four architectural layers (apart from the database management system provided by a database vendor). The four layer architecture is illustrated in [Figure 3](#).

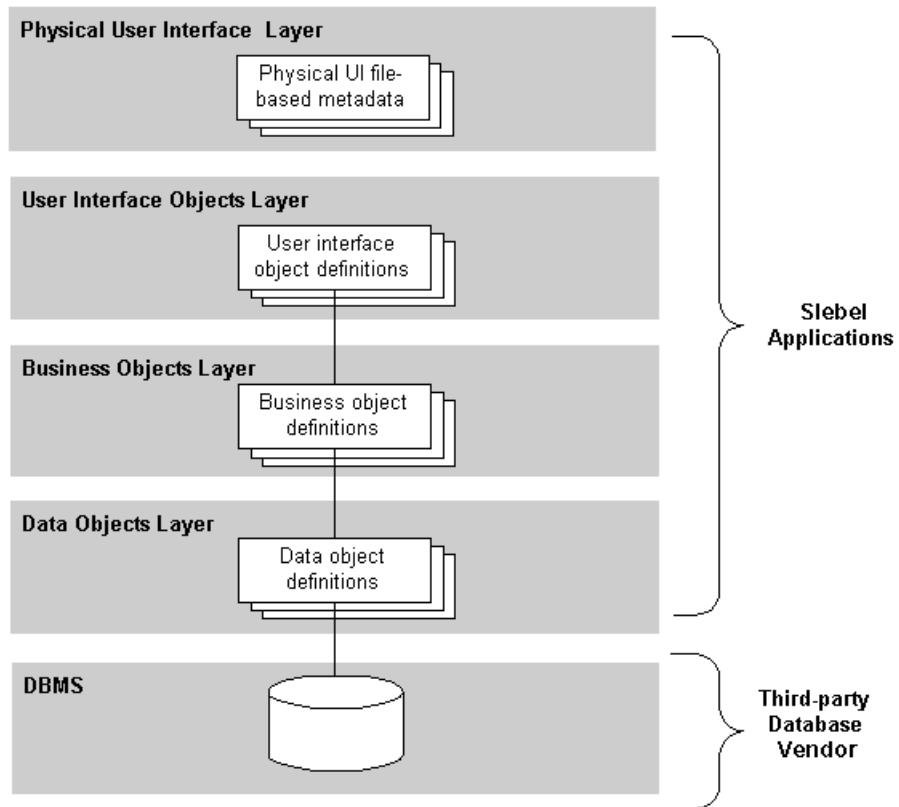


Figure 3. Architectural Layers of Object Definitions

The Physical User Interface Layer consists of the physical files, templates, style sheets, and other file-based metadata that render the UI. See [“Physical UI Layer” on page 74](#) for detailed information.

The Logical User Interface Objects layer consists of user interface object definitions that define the visual interface that the user sees and interacts with in a Web browser. Data from business object definitions is represented to the user for viewing and modification by means of user interface object definitions. See [“Logical User Interface Objects Layer” on page 68](#) for detailed information.

The Business Objects layer consists of business object definitions that are built on data object definitions, and selectively combine and associate data object definitions into logical data constructs that are useful for application design. See [“Business Objects Layer” on page 62](#) for detailed information.

The Data Objects layer consists of data object definitions that directly map the data structures from the underlying relational database into Siebel applications, providing access to those structures by object definitions in the Business Objects layer. See [“Data Objects Layer” on page 58](#) for further information.

Data Objects Layer

Data object definitions create a layer of abstraction over the DBMS, insulating the application and the developer from database administration and restructuring. The Data Objects layer is implemented by means of the data manager classes.

Object definitions in the Data Objects layer provide a logical representation of the underlying physical database (constructs like table, column, and index), and are independent of the installed DBMS.

This product feature allows you to migrate a Siebel application from one DBMS to another without having to modify the repository file where the metadata is stored. In fact, you can have a single repository file and a mix of databases from different vendors (for example, one on the server, another for connected clients, and a third for remote clients).

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

Relationship Between the Siebel Application and the Database

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

The layered architecture in Siebel applications places various object definitions at different levels relative to the database. In immediate proximity to the database are the data object definitions. Individual data object definitions directly represent individual tables, columns, and indexes in the DBMS. At an intermediate level of relationship are the business object definitions. All business object definitions are based on data object definitions or other business object definitions; they do not have any direct relationship to the database. The object definitions most indirectly related to the database are the Web interface object definitions. These are based on business object definitions, and do not directly access either the database or the data object definitions.

A developer or customer can change or extend Siebel applications without impacting the database in the DBMS. Similarly, you can make database changes without impacting, or only minimally impacting, the applications. This is possible because each architectural layer insulates the layers above it from the layers beneath, including the DBMS at the bottom of the architecture.

For example, a form applet is a Web interface object definition that implements a data entry form. The textbox controls in which the user enters data for individual columns in the database are tied not to those database columns, or to column object definitions in the Data Objects layer, but to field object definitions in the Business Objects layer. The controls represent fields, the fields represent column object definitions, and the column object definitions represent database columns.

Standard Tables and Columns

A set of standard tables is provided with Siebel applications for installation in the DBMS at a customer's site. These provide data storage for all of the features included in the standard Siebel applications, as well as expansion capability. The standard tables and their columns and indexes are represented by corresponding table, column, index, and index column object definitions.

Standard tables and their table object definitions generally are of the following types:

- **Data Tables.** The data tables comprise the bulk of the table object definitions in Siebel applications. The columns in data tables provide the data for fields. A data table may serve as a base table for a business component. That is, it may provide the primary source of data for that business component. Data tables may be provided with additional columns by means of extension tables.
- **Intersection Tables.** An intersection table (in combination with certain business object definitions) implements a many-to-many relationship between two data tables. The intersection table provides the means to implement the many-to-many relationship as two one-to-many relationships, which the underlying DBMS is equipped to handle (there is no database construct that implements many-to-many relationships directly).
- **Extension Tables.** An extension table provides additional columns that cannot be directly added to a data table. The database product may support only a limited number of columns, or will not allow adding a column to a table once it is populated with data. An extension table allows you to provide additional columns for use as fields in a business component without violating DBMS or Siebel restrictions.
- **Interface Tables.** The interface tables are intermediate database tables between the Siebel application database and other databases. Interface tables are used by the Siebel Enterprise Integration Manager to import initial data for populating one or more base tables and, subsequently, to exchange data between Siebel applications and other enterprise applications.

The columns in standard tables are of the following types:

- **Data Columns.** The data columns are part of the original set of columns implemented in the standard Siebel applications. They hold data that is made available through fields to developers and users.

- **Extension Columns.** An extension column is a column that is not used by standard Siebel applications. However, it is included for use in reconfigured applications. There are three kinds of extension columns:
 - **Standard Extension Columns.** Standard extension columns are included in standard extension tables for developer use. They are named ATTRIB_nn, where nn is a value between 01 and 47 (for example, ATTRIB_13). Each has a predefined data type and length. (Note that some tables—for example, S_ORDER_X, SPROD_INT_X—have more than 50 extension columns, and that adding more columns may affect application performance adversely. Also note that some extension columns are used by the Siebel application and are unavailable to developers.)
 - **Custom Extension Columns in an Extension or Intersection Table.** These are columns added by the developer to an extension or intersection table, and given a custom name.
 - **Custom Extension Columns in a Base Table.** These are columns added by the developer to the base table of an extension table. Their names have the X_ prefix.
- **System Columns.** Various system columns appear in all tables in Siebel applications, although no one set of system columns appears in every table. Some standard system columns are ROW_ID, CREATED, and CREATED_BY. The developer may use the data in system columns for various purposes; for example, the ROW_ID column in tables is used in the construction of joins. Generally the developer does not modify the data in system columns, although there are exceptions, such as certain of the system columns in interface tables.

The set of tables and columns provided in standard Siebel applications implements a very comprehensive design, and supports a very wide range of configuration activities. However, it is also designed to work in certain ways, and changes made that are not in accordance with Siebel standards can slow performance or cause software failures. A developer can add extension tables as well as extension columns to base tables using Siebel Tools. The developer cannot, however, add new base tables, delete base tables and columns, or modify the properties of base columns. Rather, the extension table and extension column facilities are employed.

Standards, limitations, and procedures for using extension tables are discussed in [“Extension Tables” on page 196](#). For information on database enhancement, see [Chapter 6, “Adding Custom Extensions to the Data Model.”](#)

Data Object Types

The object types for the data object layer are described briefly below. For more detailed descriptions, see *Object Types Reference*.

- **Table.** A table object definition is the direct representation of a database table in a DBMS. It has column and index child object definitions that represent the table's columns and indexes. Table, column, and index object definitions within Siebel Tools provide a detailed picture of all of the tables, columns, and indexes in use in the DBMS.
- **Column.** A column object definition represents one column in the database table. Database columns in a database table are represented by the column object definitions that are children of the corresponding table object definition. Each column in the table has a corresponding column object definition.
- **Index.** Each index object definition identifies a physical index file in the DBMS.

Business Objects Layer

The Business Objects layer consists of business object definitions that are built on data object definitions and selectively combine and associate data object definitions into logical data constructs that are useful for application design. The Business Objects layer in the repository is implemented by means of the object manager classes.

The major object types in the Business Objects Layer are:

- **Business component.** One fundamental business entity in the enterprise—for example, Contact, Business Address, or Activity.
- **Business object.** Denotes a functional area that is a grouping of one or more related business components.

Business Components

A business component consists of multiple fields that characterize it; for example, some of the fields for Contact might consist of first name, job title, and email address.

Figure 4 shows that a business component maps to one main table in the Data Objects layer and fields in the business component map to columns in the main table. In this example, the Contact business component maps to the S_CONTACT table.

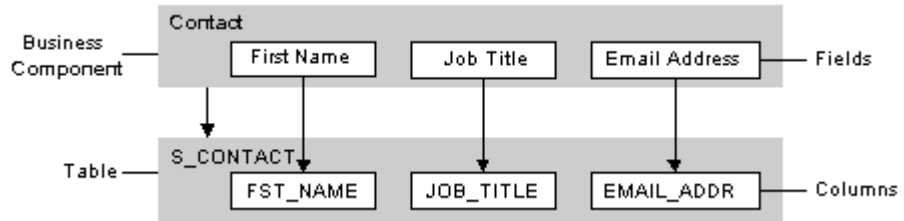


Figure 4. Business Component Mapping to the Main Contact Table

Business components can also include data from related tables. Figure 5 shows the Contact business component mapped to the main table (S_CONTACT) and also two related tables, S_ORG_EXT and S_OPTY_CON.

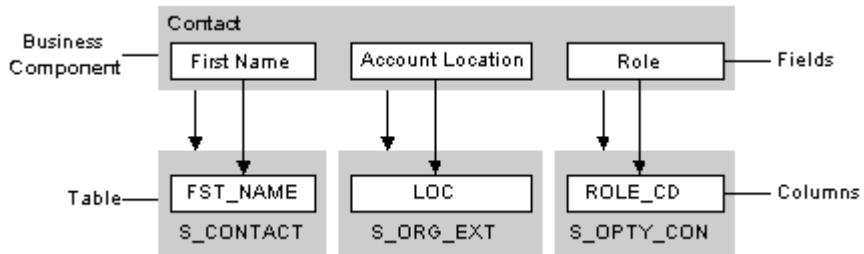


Figure 5. Business Component Mapping to Data in Related Tables

You can think of a business component as a denormalized virtual database table that spans multiple real tables.

The “grouping” of data can be achieved by:

- Referencing an extension table
- An explicit join of two tables
- A link that joins data in an intersection table

Business Objects

A business object represents a major functional area of the enterprise—every major entity has a business object. Examples of business objects are Opportunity, Account, and Contact.

A business object is a collection of related business components; for example (as shown in [Figure 6](#)), the Opportunity business object consists of Opportunities plus related Contacts, Activities, Products, and Issues. Each business object has one business component that serves as the master or driving business component. This master business component provides focus for the business object, and they both have the same name (the name is Opportunity in [Figure 6](#)).

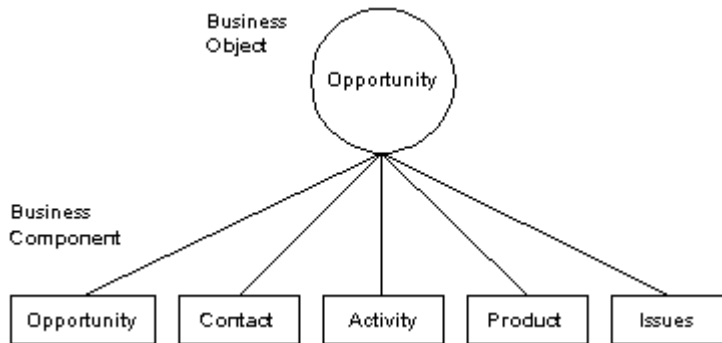


Figure 6. The Opportunity Business Object and Its Business Components

The Opportunity business component has one and only one parent, which is the Opportunity business object. A link would be required for the other business components (Contact, Activity, Product, and Issues) to connect with the Opportunity business object.

Business Component Reuse

As shown in [Figure 7](#), a business component can be defined once in terms of a logical collection of columns from one or more tables, and then used in many different business object contexts. One specific area of application configuration where reuse plays a significant role is virtual business components. For more information, see *Integration Platform Technologies: Siebel eBusiness Application Integration Volume II*.

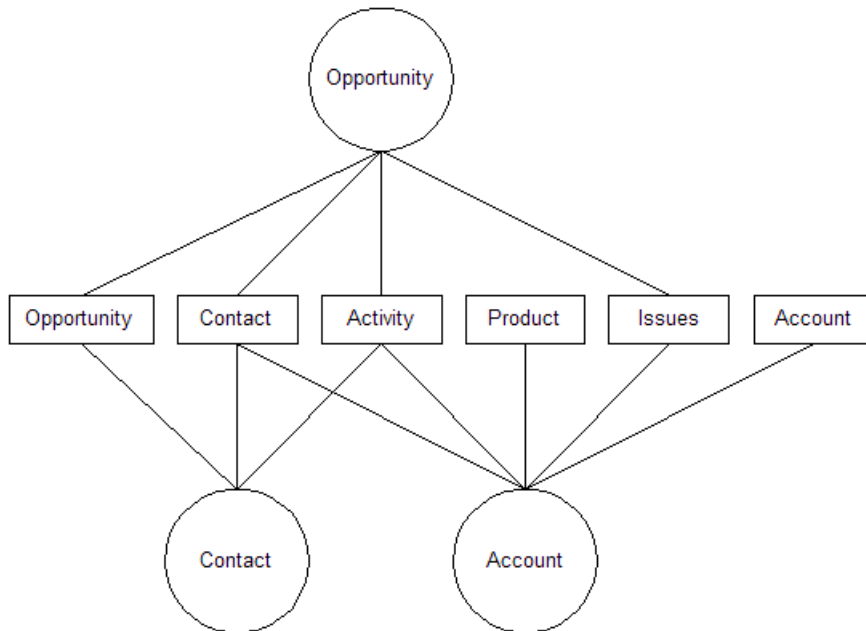


Figure 7. Business Component Reuse

Business Object Types

The object types for the business object layer are described briefly below. For more detailed descriptions, see *Object Types Reference*.

- **Business Component.** A business component is a logical entity that associates columns from one or more tables into a single structure. Business components provide a layer of wrapping over tables, causing applets to reference business components rather than the underlying tables. This design creates convenience (all associated columns together in one bundle), developer-friendly naming, and the isolation of the developer role from the database administrator role.

Multiple users can instantiate copies of the same business component. Data changes made by any one user are reflected in all instances of the business component.

- **Field.** A field object definition associates a column to a business component. This is how columns from tables are assigned to a business component and provided with meaningful names that the customer developer can easily change. Alternately, a field's values can be calculated from the values in other fields in the business component. Fields supply data to controls and list columns in the Web interface.
- **Business Object.** A business object implements a business model (logical database diagram), tying together a set of interrelated business components using links. The links provide the one-to-many relationships that govern how the business components interrelate in the context of this business object.

NOTE: The object type called Business Object is not to be confused with the general category called business object types. Business Object is one of the object types in the Business Objects layer. Similarly a business object, which is one kind of object definition, is not the same as the category “business object definitions.”

- **Business Object Component.** A business object component object definition is used to include a business component and, generally, a link in the business object. The link specifies how the business component is related to another business component in the context of the same business object.
- **Link.** A link implements a one-to-many relationship between business components. The Link object type makes master-detail views possible. A master-detail view displays one record of the master business component with many detail business component records corresponding to the master. A pair of links also may be used to implement a many-to-many relationship.

- **Multi-Value Link.** A multi-value link is used in the implementation of a multi-value group. A multi-value group is a user-maintainable list of detail records associated with a master record. The user invokes the list of detail records from the master record when it is displayed in a list or form applet. For example, in an applet displaying the Account business component, the user can click the Select button to the right of the Address text box to see a pop-up window displaying multiple Address records associated with the currently displayed account.
- **Join.** A join object definition creates a relationship between a business component and a table that is not the business component's base table. The join allows the business component to build fields using columns from the non-base (joined) table. The join uses a foreign key in the business component to obtain rows on a one-to-one basis from the joined table, even though the two do not necessarily have a one-to-one relationship.
- **Join Specification.** A join specification is a child object type of join that provides details about how the join is implemented within the business component.
- **User Property.** A user property is a temporary storage field in a business component that is not tied to the database. When a user sets a value in a business component user property, it is not visible to other users of the same business component. User properties are not saved across sessions.
- **Business Service.** A business service is a reusable module containing a set of methods. It provides the ability to call its C++ or script methods from customer-defined scripts and object interface logic, through the invoke-method mechanism.

For more information, see *Integration Platform Technologies: Siebel eBusiness Application Integration Volume II*.

Logical User Interface Objects Layer

The Logical User Interface Objects layer consists of user interface object definitions that determine the visual interface that the user sees and interacts with in a Web browser. Data from business object definitions is represented to the user for viewing and modification by means of user interface object definitions. The Applet user interface object type (along with its child object types, particularly Applet Web Template) implements application units consisting of data controls, editable scrolling list tables, business graphics, and so on. Other user interface object types control toolbar and menu implementation, and the visual grouping of applets on screen.

The user interface defines the visual elements with which users interact—for example:

- Layout of applets and views
- Navigation
- User interface controls (for example, buttons and check boxes)

The User Interface Objects Layer is insulated from the Data Objects Layer and the underlying database by the Business Objects Layer.

Various interface elements in a Siebel eBusiness Application are shown in [Figure 8](#).

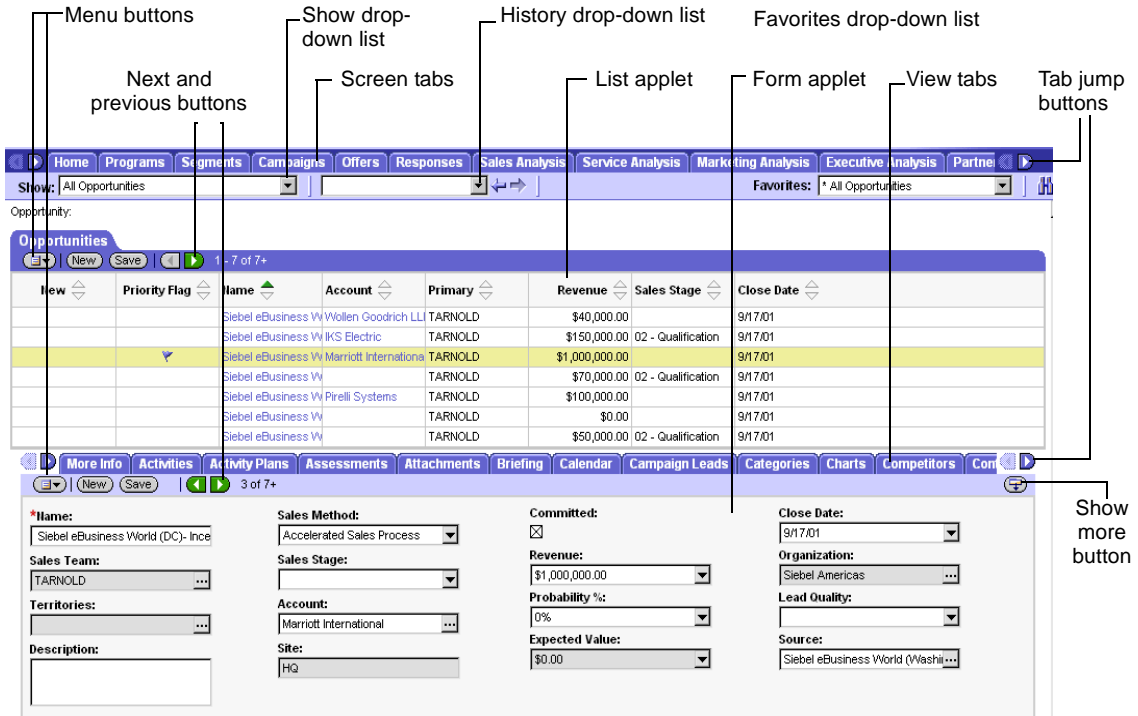


Figure 8. User Interface Objects in a Siebel eBusiness Application

Applets

An applet allows access to the data of a single business component for viewing, editing, and modifying fields in the business component.

An applet can be configured to allow data entry for a single record, to provide a scrolling table displaying multiple records, or to display business graphics or a navigation tree.

As shown in [Figure 9](#), an applet consists of controls that map to fields in the underlying business component or simply support user interactions with the application, as is the case with buttons, for example:

- A list column for a list applet (the top applet in [Figure 9](#))
- A text box for form applets (the bottom applet in [Figure 9](#))

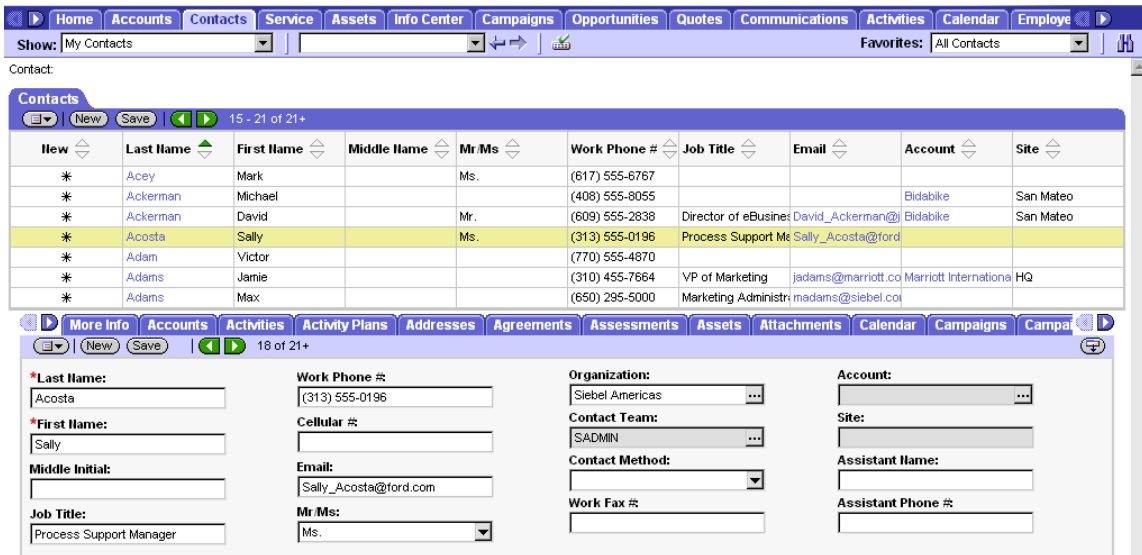


Figure 9. Fields in a Contact Applet Mapped to the Contact Business Component

All data in a given applet must be part of the business component that calls it.

Data in a given business component can be from a single table or multiple tables.

Business components can be reused in multiple applets. Several applets can reference the same business component definition. This is a benefit of having a single definition in the business component of the logical-to-physical relationship. Since the user interface (applet) layer is abstracted from the physical layer using the business component, there is no need to revisit the details of multiple tables in the physical layer for each of the applets that will display data for the same business function.

NOTE: Do not confuse the applet construct in Siebel applications with Java applets. They are somewhat similar, but there are significant differences.

Views

A view presents one or more applets together at one time in a predefined visual arrangement and logical data relationship. Views are named, and a specific view is selected by name from a combination of menus or tab symbols.

A given view is mapped to a single business object, which determines the relationship between data displayed in two or more applets in the view. For example, the Opportunity Contacts Detail view is based on the Opportunity business object. Opportunity data is always displayed as the master or parent, and other types of data (for example, Contacts) are displayed as child records of a particular Opportunity. This allows users to see all the contacts associated with a single Opportunity.

Each applet in a view must map to a business component in that business object.

The business components that are required to be included in each view are:

- For a view based on the business object, all business components to which applets in that view are mapped
- Any business component whose data is exported in a report from a view based on the business object

NOTE: The business component you include can be a child of another business object.

Screens

A screen is a collection of related views.

Screens are associated with major functional areas of the enterprise.

In general, all views in a screen map to the same business object—an exception is administration screens.

Applications

An application (for example, Siebel Sales) is a collection of screens.

NOTE: Do not confuse Siebel applications with mobile Siebel application executables (that is, .exe files).

You can access Siebel applications through the Siebel Web client, dedicated Web client, mobile Web client, handheld, or wireless Web client.

Your organization may have licensed more than one Siebel application (for example, Siebel Sales and Siebel Call Center), to be used by different groups (for example, the sales team and the customer support team) within your organization.

In that situation you can install the Siebel Sales and Siebel Call Center as separate applications, or as a single application in which you establish different views for the relevant responsibilities of the two groups using them.

Logical UI Object Types

The object types for the logical UI object layer are described briefly below. For more detailed descriptions, see [Chapter 5, “Data Objects Layer”](#) or *Object Types Reference*.

- **Application.** An application is a collection of screens. The application is opened in a Web browser on the user's desktop by attaching to a specified URL. The screens are accessed from the tab bar and the Site Map, as defined in the application. Siebel eService is an example of an application. Each combination of screens that is appropriate to a specific class of users can be provided as an application.
- **Page Tab.** A page tab object definition associates a screen to the page tab's parent application object definition and includes it as a tab in the tab bar.
- **Screen Menu Item.** A screen menu item object definition associates a screen to the application and includes the screen as a menu item in the Site Map.
- **Screen.** A screen is a logical collection of views. It is not a visual construct in itself; rather, it is a collection of views that the menu bar and view bar can display. The active screen is selected from the Site Map or the tab bar.

- **Screen View.** A screen view object definition associates a view with the screen view's parent screen object definition. This is how views are included in screens.
- **View.** A view is a collection of applets which appear on screen at the same time. A view can be thought of as a single window's worth of related data forms (applets). The Siebel application window displays one view at any one time. The user can select the current (active) view from the second-level navigation tab or from the Site Map. A view is associated with the data and relationships in a single business object.
- **Applet.** An applet is a form, composed of controls, that occupies a portion of the Siebel application window. An applet can be configured to allow data entry, provide a scrolling table of business component records, or display business graphics, a navigation tree, or a similar user interface unit. It provides viewing, entry, modification, and navigation capabilities for data in one business component. Pop-up windows for multi-value groups and record selection are also implemented as applets.
- **Control.** One control object definition corresponds to one data control in a form applet, such as a text box, check box or command button. A control is something in the applet with which the user can interact. A control usually either exposes data from one field in the business component, or invokes programming logic (in the case of a PushButton control).
- **List.** List is a child object type of Applet. A list object definition specifies property values that pertain to the entire scrolling list table and provides a parent object definition for a set of list columns.
- **List Column.** A list column object definition corresponds to one column in the scrolling list table in a list applet, and to one field in the business component.
- **Web Template, Applet Web Template, View Web Template.** Identify external HTML (or other markup language) files that define the layout and Siebel Web Engine interactions for an applet or view.
- **Applet Web Template Item.** Defines a control, list item, or special Web control in the Web implementation of an applet.
- **View Web Template Item.** Defines the inclusion of an applet in the Web implementation of a view.

Physical UI Layer

The physical user interface (UI) consists of the physical files, templates, Siebel tags, style sheets, and other file-based metadata that control the layout (as opposed to the content) of the user interface (for example, CSS, SWF, and GIF files). The Applet Web Template, Applet Web Template Item, View Web Template, View Web Template Item objects are part of the logical UI layer; their object definitions are stored in the Siebel repository. Examples of physical UI objects are:

- **Templates files.** A Siebel template is a special kind of HTML file that defines the layout and formatting of elements of the user interface (such as views, applets, and controls). It provides this layout information to the Siebel Web Engine when rendering Siebel objects in the repository to HTML files.

The layout and style of HTML Web pages is dynamic, which allows simultaneous support for multiple browser types and versions. This is accomplished through the conditional branching in Web templates.

- **Tags.** Siebel tags are special tags you insert into template files. They specify how Siebel objects defined in the repository should be laid out and formatted in the final HTML page in the user's Web browser.

The process of configuring a Web application separates the layout and formatting from the application definition and binding to data. You use Siebel tags to map objects into a HTML physical layout.

- **CSS.** Siebel Cascading Style Sheets are external style sheet documents (of type text/CSS) to define how HTML or XML elements and their contents should appear in a Web document.

CSS provide rules for resolving conflicts in HTML or XML These rules consist of two main parts: a selector and a declaration. The declaration has two parts: property and value.

Summary of the Major Object Types

Figure 10 shows the major object types in a Siebel application, and the relationship between them.

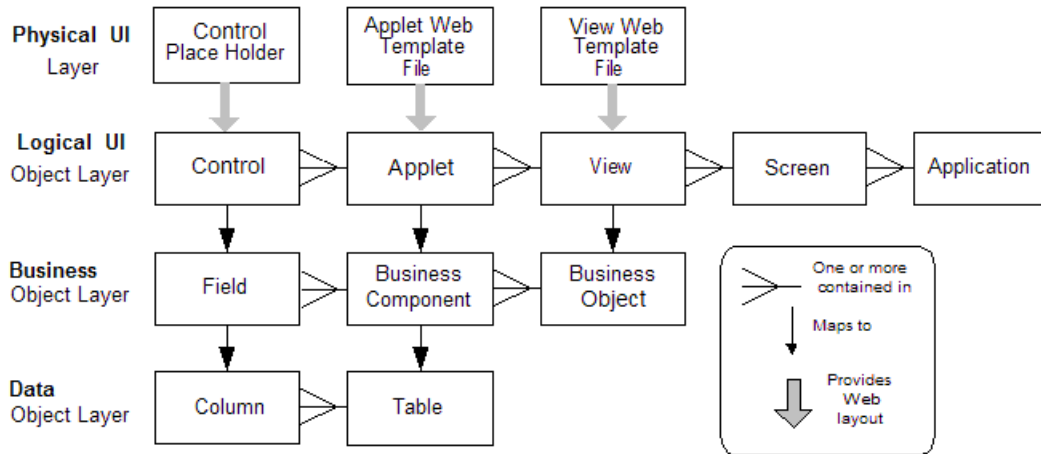


Figure 10. Overview of the Major Object Types and Their Relationships

Operating Architecture Overview

The client-server architecture of the standard HTML client is illustrated in [Figure 11](#).

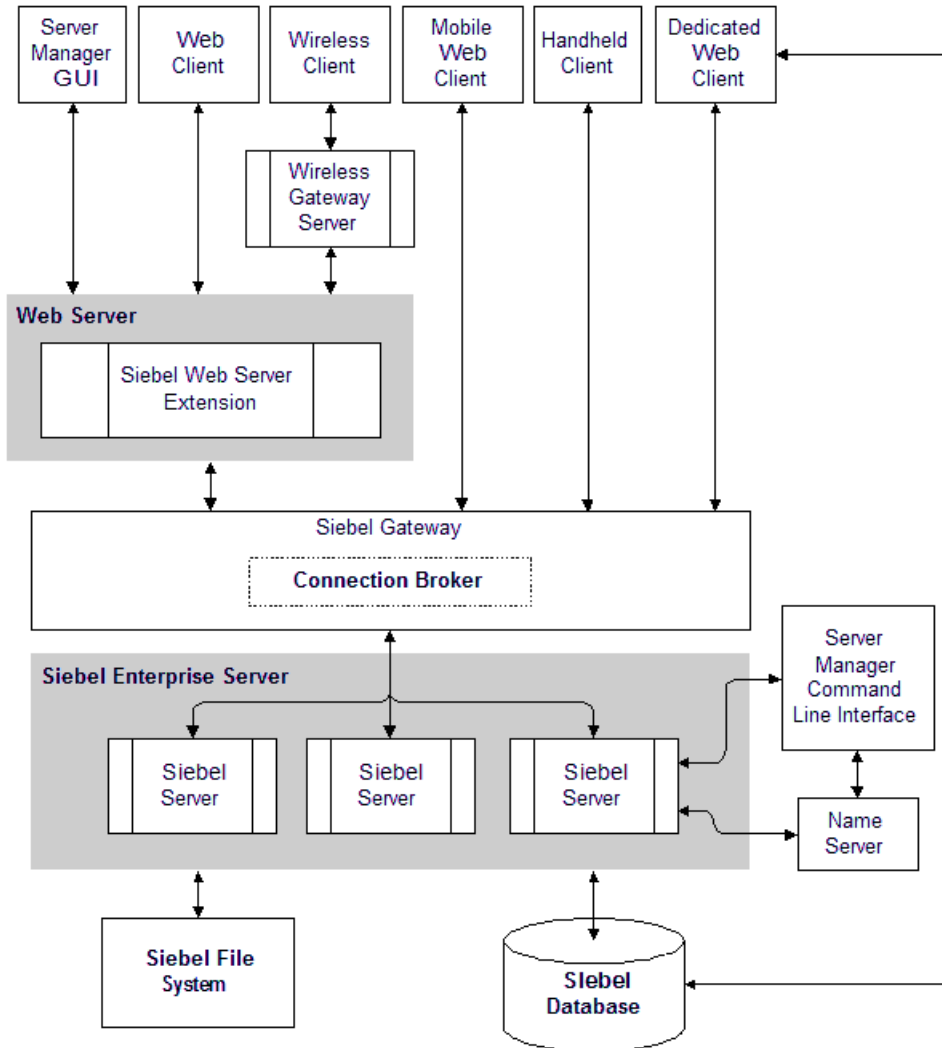


Figure 11. Client-Server Architecture for Siebel Web Engine

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

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

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

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

- **Siebel Web Engine.** This is also called SWE. The user interface in Siebel applications is generated as HTML pages on the server, and is passed to an unmodified Web browser through HTTP. The Siebel Web Engine (in combination with the Siebel plug-in on the Web server) makes possible the deployment of these applications. A Web browser client (or other Siebel client) interacts with the server-based object manager through the Siebel Web Engine, providing the means for the user to view and edit data. SWE retrieves and updates data by interfacing with the object manager. A notification mechanism between SWE and the object manager is used so that when one applet modifies data in any business component, all other applets are notified immediately so that they can update their data on the screen.
- **Data Manager.** The data manager is a facility inside the object manager that issues SQL queries in response to object manager requests, and passes back database result sets to the object manager. The data manager is composed of one connector DLL for each type of database connection supported by the system. The object manager dynamically loads the appropriate DLL based upon the required data source.

Siebel Web Engine Infrastructure

This section describes the Siebel Web Engine (SWE) architecture and functionality in greater detail.

The Siebel Web Engine makes possible the deployment of applications in HTML and other markup languages. A Web browser client (or other Siebel client) interacts with the server-based object manager through the Siebel Web Engine, as shown in [Figure 13 on page 82](#).

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

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

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

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

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

How the Siebel Web Engine Generates the Web Application

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

The diagram in [Figure 12](#) provides a graphical depiction of the relationships between the various objects in a Web application.

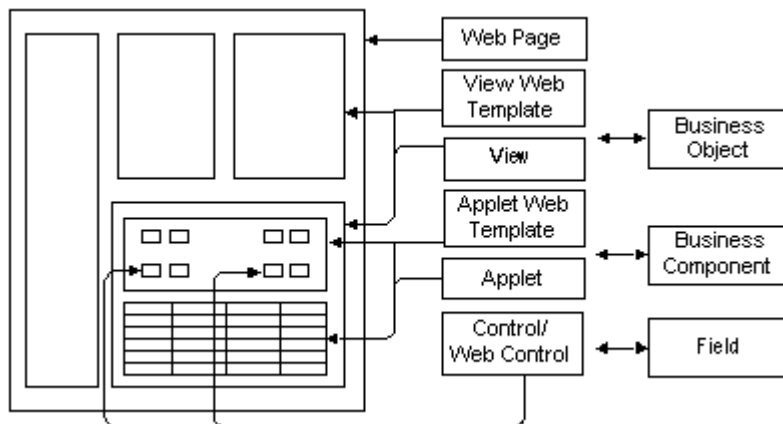


Figure 12. Relationship Between Objects in a Web Application

Running the Web Application

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

- 1 Reads the repository definition of the application.
- 2 Retrieves relevant data from the database through the Application Object Manager.

- 3 Retrieves the repository object definition of the view and applet to display the data within it.
- 4 Reads the .SWT file and maps the retrieved data and applet and view information to the corresponding placeholders in the .SWT file.
- 5 Delivers the HTML page (including the standard HTML and the retrieved data) through the Web Server, back to the user's browser for display as a Web page.

About Standard and High Interactivity

Siebel applications are deployed in either Standard Interactivity or High Interactivity mode.

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

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

- Browser scripting.
- Implicit commit, which allows the application to automatically save a record when the user steps off it.
- User interface features, such as drag and drop column reordering, drag and drop file attachments, keyboard shortcuts, smart controls for calendar, calculator and currency functions, and applet scroll bars.

NOTE: Partner applications can be deployed in either Standard Interactivity or High Interactivity mode.

One way to distinguish between Standard and High Interactivity modes is by the underlying technologies used by each mode (see [Table 5](#)).

Table 5. Technology Differences Between Standard and High Interactivity

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

There are requirements for deploying in both modes. For example, Standard Interactivity requires that browsers support HTML 3.2 and JavaScript interpretation. High Interactivity mode requires support for additional technologies, such as Java Virtual Machine or Java Runtime Environment, and Active X controls.

For specific requirements for each mode, see *Siebel System Requirements and Supported Platforms*.

For specific requirements related to browser settings, see *Siebel Web Client Administration Guide*.

JavaScript Object Architecture in High Interactivity

Extension of browser behavior is accomplished by means of JavaScript, an interpreted language running in many Web browsers, with different support for the standard and varying extensions.

The browser objects layer allows you to add scripts, which run in the browser, to the traditional Siebel objects. See [“Application Enhancement Through Scripting and Object Interfaces” on page 171](#).

Objects representing the applet, business component, business services, and application objects live in the browser address space as JavaScript objects, and provide communication with the server. These object types are the same object type instantiated within the browser: browser applet, browser buscomp, browser business service and browser application. Initially, these pass through to the SWE, but can become more sophisticated and provide caching and other local processing.

You can script instances of browser applets, browser buscomps, browser business services, and browser applications.

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

In the diagram in [Figure 13](#), the different boxes represent different components or different parts of the application. Specialized business component logic is shared among all platforms; specialized Web applet logic is shared between all HTML clients; and browser logic is the only part that is browser-specific.

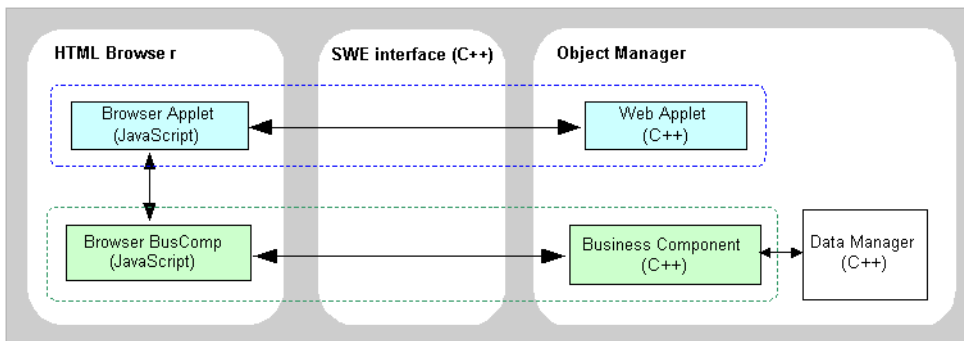


Figure 13. JavaScript Architecture for High Interactivity

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

Enabling and Disabling High Interactivity for Applications

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

NOTE: High Interactivity is not supported for customer applications.

To enable or disable high interactivity

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

```
[SWE]  
  
HighInteractivity=TRUE
```
 - To disable High Interactivity (run Standard Interactivity mode), delete the *HighInteractivity* parameter from the .cfg file.
- 4 Save and close the .cfg file.

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

Enabling and Disabling High Interactivity for Views

You can control whether views appear in High Interactivity mode by the underlying class definitions of the applets that appear on the view. A view is displayed in High Interactivity mode when the underlying classes of all the applets in the view have the High Interactivity Enabled property set to 2, 3, 4, or 5. The possible values for the High Interactivity Enabled property are summarized in [Table 6](#).

Table 6. High Interactivity Enabled Property Values

Value	Works with High Interactivity	Works with Standard Interactivity	Cachable
1	No	Yes	No
2	Yes	No	Yes
3	Yes	No	No
4	Yes	Yes	Yes
5	Yes	Yes	No

High Interactivity Configuration Considerations

When configuring applications for deployment in High Interactivity mode, consider the following:

- Browser scripting is fully supported in High Interactivity mode.
- For fields to interpret and display custom HTML, such as a URL entered by the user, the field's Type property must be set to URL. If it is not set to URL, the HTML is presented and interpreted as plain text. For example, if a user typed a URL in a field of type TEXT, the URL would not be recognized as a link to a Web page.
- You cannot modify the appearance of the rich text editor.
- You cannot modify the background and text color of list applets.
- You cannot place method-invoking controls, such as the delete function, on every row in a list. Instead place a button that calls the method on the applet itself. The function will act on the selected record.

- There are cases when an application's configuration file is set to run in High Interactivity mode and all the applets in a view are configured to support High Interactivity, but the view appears in Standard Interactivity mode. Reasons this might occur are:
 - One of the applets is in the Query mode. Because High Interactivity implicitly supports query operations from the user interface, it does not support the explicit use of the Query mode.
 - One of the applets is in the New mode and uses a New template that is different from the Edit template used in its default mode. This can be avoided by inactivating New templates associated with the applets used in High Interactivity applications. The framework will then default to using the Edit template itself to create new records.
 - One of the list applets has multi-row edits or multi-row select enabled.
 - One of the list applets is a hierarchical list applet.
 - The view uses a template that shows applets in a catalog-style layout. None of the employee applications should be using this layout.
 - A combo box picklist uses Long Lists or has an associated pick applet. For example, if you perform an action from a High Interactivity applet that causes a pick applet to be displayed, the pick applet will not be in High Interactivity mode.

Integrating Siebel with J2EE

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

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

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

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

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

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

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

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

Siebel Partner Connect and Siebel Tools for Partner Connect

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

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

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

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

This chapter describes the basics of working with Siebel Tools. It describes the main windows of the Siebel Tools user interface and basic tasks, such as viewing, creating, and modifying object definitions.

What Is Siebel Tools?

Siebel Tools is an integrated environment for configuring all aspects of a Siebel application so a single configuration can be:

- Deployed across HTML and wireless clients
- Used to support multiple Siebel applications and languages
- Easily maintained
- Automatically upgraded to future Siebel product releases

Siebel Tools is not a programming environment; it is a declarative application configuration tool. Standard Siebel applications provide a core set of object definitions that you can use as a basis for your tailored application. Using Siebel Tools and other configuration tools that are part of a Siebel solution, Siebel application developers, system administrators, and database administrators can customize a standard Siebel application without modifying source code or SQL. Some of the configuration tools are accessed through the Siebel applications. Siebel Tools, however, is a separate product with its own user interface.

Navigation in Siebel Tools is done mainly in two windows:

- Object Explorer window
- Object List Editor window

The Object Explorer employs a hierarchical tree-structure user interface (similar to that of the Microsoft Windows Explorer) to allow you to browse the object types that are stored in the Siebel Repository.

Other Siebel Tools windows, like the Object List Editor and Properties windows, show you detail about individual objects in the Siebel repository.

Siebel Tools Features

The main features of Siebel Tools include:

- [“Siebel Tools Application Window”](#)
- [“Siebel Objects”](#)
- [“Siebel Object Explorer” on page 91](#)
- [“Web Layout Editors” on page 91](#)
- [“Script Editors” on page 92](#)
- [“Target Browser Support” on page 97](#)
- [“Object Repository” on page 98](#)

Siebel Tools Application Window

The main application window that appears when you start Siebel Tools is the Siebel Tools Object Explorer. For more information about this window, see [“About the Object Explorer” on page 98](#).

Siebel Objects

Siebel applications are built on object-oriented principles. A Siebel object definition is a data construct in the repository file that defines an element of the:

- User interface
- Business entities
- Database organization

For a discussion of objects, object types, and objects definitions, see [Chapter 2, “Siebel Architecture \(Basic Concepts\).”](#)

In general, a large portion of the task of configuring a Siebel application involves modifying or creating object definitions within Siebel Tools.

Siebel Object Explorer

The Siebel Object Explorer gives you the ability to customize Siebel products. It is a graphical editing tool used for modifying and managing object definitions. Siebel Object Explorer is the main application window and appears when you start Siebel Tools. (For more information about using the Siebel Object Explorer, see [“About the Object Explorer” on page 98.](#))

Web Layout Editors

The Web Layout Editor allows you to:

- Add and map controls and list columns to applet Web templates. You can preview applets as they would be rendered at runtime.

See [“Editing the Web Layout of Applets” on page 544](#) for more information.
- Modify existing views and construct new ones by dragging and dropping applets onto the view layout window. You can view list and form applets and the container page in the Preview mode. No additional specification or code is required for defining the relationships between the applets.

See [“Editing the Web Layout of Views” on page 566](#) for more information.
- Add and delete controls from Web page templates, modify control properties, and map controls to placeholders. You can also preview Web pages as they would appear at runtime.

See [“About Web Pages” on page 579](#) for more information.
- Visually edit Siebel application menu structures. It is accessed by right-clicking an object in the Object List Editor and selecting Edit Web Menus.

You can launch the Web Layout Editor directly from an applet, view, or Web page in the Object List Editor by right-clicking and choosing Edit Web Layout.

Siebel Applet Editor and Siebel View Editor are editors used to configure Windows clients. If you are using earlier versions of Siebel applications, you may wish to use these editors as part of the migration of their configurations to the Siebel 7.0 Web client. Siebel Applet Editor and Siebel View Editor do not show by default. To show them, you the `ClientConfigurationMode` parameter in the `tools.cfg` file to `ALL`. Note that these editors will not display pre-7.0 chart applets.

NOTE: When working with object definitions in make sure that the Application drop down list in the Configuration Context Toolbar is set to the correct Siebel application. Specifying the correct application will make sure that the Web Layout as seen in Siebel Tools will be consistent to UI of the Siebel vertical application.

Script Editors

Scripting is used to implement functionality that cannot be achieved declaratively (that is, by changing object properties). The Server Script Editor and the Browser Script Editor are used to add scripts to Siebel objects. Scripting is supported through three features in Siebel applications. These are Siebel VB, Siebel eScript, and Browser JavaScript. For information on scripting, see [“Application Enhancement Through Scripting and Object Interfaces”](#) on page 171.

Server Script Editor

The Server Script Editor is used to create and modify Siebel VB and eScript scripts. The Edit Server Scripts functionality is accessed by right-clicking a scriptable entry in the Object List Editor or from the View > Editors > Server Script Editor menu. See *Siebel eScript Language Reference* and *Siebel VB Language Reference* for further details.

Browser Script Editor

The Browser Script Editor allows you to write and edit JavaScript that runs within the client browser. The Edit Browser Scripts options is available by right-clicking Applet, Business Component, Application, and Business Service object definitions in Tools. You can also access it from View > Editors > Browser Script Editor menu. The result takes the form of Applet Browser Script, BusComp Browser Script, and so on. Each of these object types has a set of scriptable events, generally including InvokeMethod and PreInvoke Method, as well as object-type specific ones, such as ChangeRecord and ChangeFieldValue for applets. See *Siebel Object Interfaces Reference* for further details and for a list of scriptable events and callable methods on browser objects.

Wizards

Various wizards in Siebel Tools step you through the process of creating and configuring an object definition of a particular object type and style—for example, a Siebel applet.

Wherever a wizard exists for a particular task in Siebel Tools, you can choose to use it or not:

- If you do not use the wizard, you can create and change property settings for object definitions directly in the Object List Editor. (Dock objects are the exception.)
- If you use the wizard, it asks you for your preferences, and then bases property settings on them.

The wizards available in Siebel Tools are shown in [Table 7](#).

Table 7. Wizards Available in Siebel Tools

Wizard Type	For More Information, See This Chapter
Form Applet	Chapter 10, “Logical User Interface Objects Configuration”
List Applet	
Chart Applet	Chapter 13, “Special-Purpose Applets and Controls”
Tree Applet	

Table 7. Wizards Available in Siebel Tools

BusComp	Chapter 8, “Defining Business Objects and Business Components”
MVG Applet	Chapter 12, “Multi-Value Group and Association Applets”
Multi Value Group	
Pick Applet	Chapter 11, “Pick Applets and Static Picklists”
Picklist	
Web Layout	Chapter 10, “Logical User Interface Objects Configuration”
View	
Applet Method Menu	
Command	
Table	Chapter 6, “Adding Custom Extensions to the Data Model”
EIM Table Mapping	
Dock Object	
External Schema Import	
Configuration Utility	Chapter 6, “Adding Custom Extensions to the Data Model”
	Chapter 17, “Repositories”
	The upgrade guide for the operating system you are using.
	<i>Siebel Server Administration Guide</i>
Upgrade	The upgrade guide for the operating system you are using.
Report	<i>Siebel Reports Guide</i>
OLEDDB	<i>Integration Platform Technologies: Siebel eBusiness Application Integration Volume II</i>
Integration Object	<i>Overview: Siebel eBusiness Application Integration Volume I</i>
Import	Chapter 17, “Repositories”

To access the new object wizards

- 1 Choose File > New Object from the Siebel Tools main menu. Icons for all the new object wizards appear in the New Object dialog box (shown in the figure below). You choose either the General, Applets, or EAI (Enterprise Application Integration) tab to access the appropriate wizard.

NOTE: Some specialized wizards are not accessible from this dialog box (for example, the Application Wizard, the Web Client Migration Wizard, the EIM Table Mapping Wizard, and the Docking Wizard).



- 2 When you click on the type of new object you want to create, the Siebel wizard guides you through the task of entering the properties that type of object requires.

Figure 14 shows the Siebel List Applet Wizard.

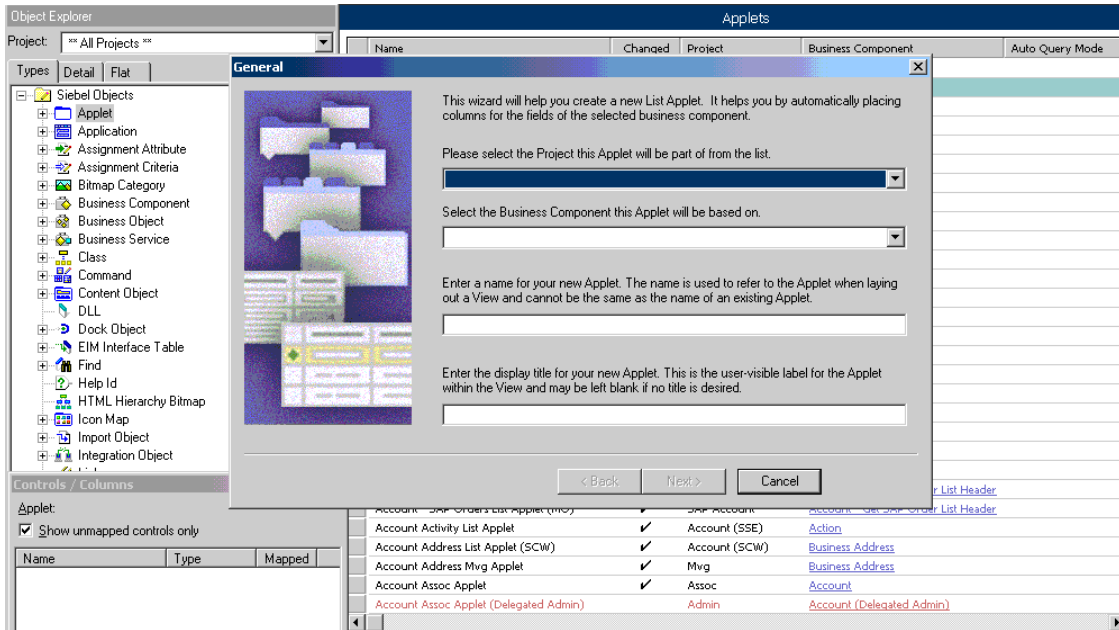


Figure 14. Siebel List Applet Wizard

NOTE: To access the EIM Table Mapping Wizard and the Dock Object Wizard, right-click entries in the Object List Editor as explained in [Chapter 6, “Adding Custom Extensions to the Data Model.”](#) You access the Web Migration Wizard from Tools > Upgrade > Web Client Migration menu.

Target Browser Support

A Web browser has defined capabilities such as cookies, table support, Java applet support, and markup language. Specific browsers are aggregated into groups of similar browser types (for example, various versions of Internet Explorer are grouped as IE5). In Siebel Tools, a Web browser type is called a target browser.

The Target Browser feature of Siebel Tools allows you to configure applications conditionally for different browsers. You can create a dynamic response for each client browser type. Based on the browser group selected, conditional template tags (template placeholders) and browser scripts are rendered in a specific way.

Select the browser type from the Target Browser drop-down list in the Configuration Context Toolbar.

The Browser Script Editor is accessible from the main menu (as shown in [Figure 15](#)), or by right-clicking on a record in the Object List Editor.

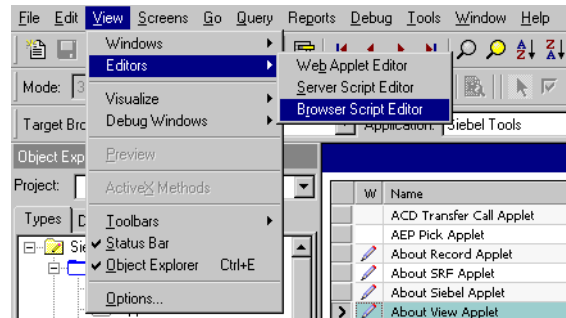


Figure 15. Accessing the Browser Script Editor

Object Repository

The Object Repository provides a multiuser development environment that includes access to check-in/check-out functionality and version control.

In a typical Siebel Tools development environment, a server repository contains the master application definition. As an application developer, you have a local repository connecting to Siebel Tools. You can check out projects on the server and copy the projects to the local database where they can be edited. After you have made the changes, you can test the application by compiling the Siebel repository file (.srf) from the local developer database. If the tests have been successful, you can perform the check-in to the server database. The check-in will copy the locked projects to the server and then unlock the projects on the server.

You can integrate the Siebel Tools check-in/check-out process with an external version control system like Microsoft Visual SourceSafe, PVCS, or ClearCase, enabling the development team to maintain a version history of all changes to the repository. For further information on this topic, see [Chapter 18, “Application Development Projects.”](#)

About the Object Explorer

[Figure 16](#) shows the windows most often visible in Siebel Tools, the Object Explorer and Object List Editor windows.

- [“Object Explorer Window”](#)
- [“Showing and Hiding Objects in the Object Explorer” on page 105](#)
- [“Object List Editor Window” on page 107](#)
- [“Other Windows” on page 108](#)
- [“Hiding the Windows” on page 113](#)
- [“Docking the Windows” on page 113](#)
- [“Image Preview” on page 114](#)
- [“Drilldown” on page 114](#)

Object Explorer Window

The Object Explorer window (see [Figure 16 on page 100](#)) shows a hierarchical representation of the major object types, and the Object List Editor window shows object definitions.

To make the Object Explorer window visible

- Choose View > Object Explorer.

By default, the Object Explorer is visible when you start Siebel Tools.

NOTE: Many of the menu commands in Siebel Tools have shortcut equivalents, which are displayed to the right of the command name in the main drop-down menus.

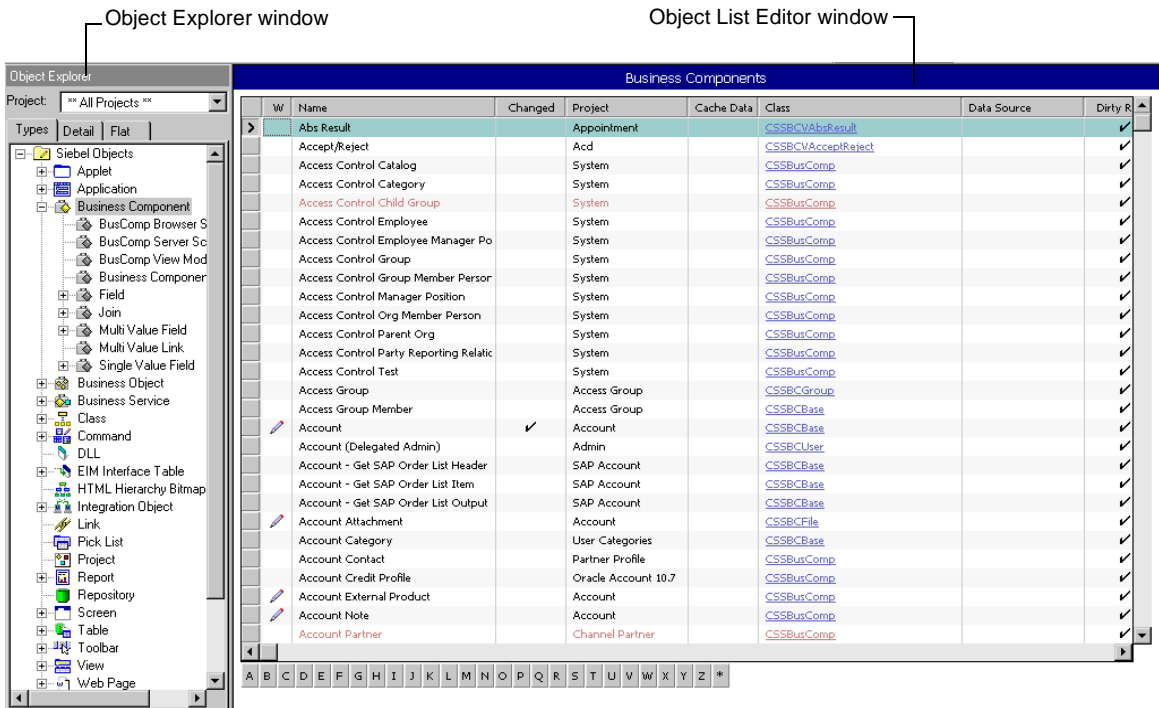


Figure 16. Object Explorer and Object List Editor Windows

The Siebel Object Explorer is composed of the following:

- A hierarchical Object Explorer that allows you to browse the various object types

- An Object List Editor for viewing and editing object definitions
- A Properties window for editing object property values
- A Windows-style search capability that allows you quickly to locate objects in the Siebel repository

Types Tab

The Types tab is selected in the Object Explorer window shown in [Figure 17 on page 102](#).

The Types tab shows all top-level object types, listed alphabetically. The Types tab shows the object hierarchy—clicking on the plus sign (+) to the left of an object type displays all the child object types of the top-level object type. Clicking on the minus sign (-) to the left of an object type collapses all its child object types.

Some object types have a hierarchy of multiple levels. For example (as shown in [Figure 17 on page 102](#)):

- One of the child object types of Applet is Control, and, at the next lowest level, Control User Properties.

- One of the child object types of Business Component is Field.

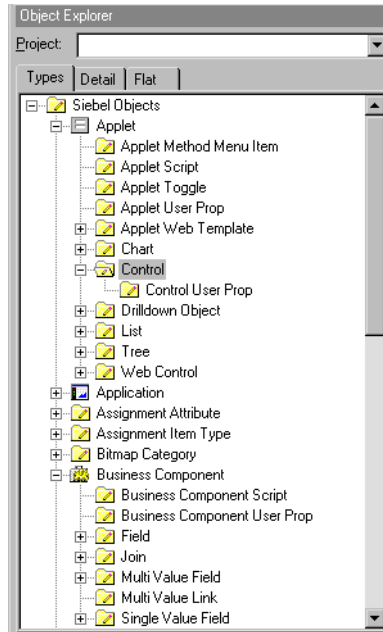


Figure 17. Hierarchy of Object Types (Types Tab)

Detail Tab

If you select the Detail tab of the Object Explorer (as shown in [Figure 18](#)) and select an object type, the Object List Editor displays all object definitions of that type in the Explorer itself. For example, [Figure 18](#) shows 24 Applications object definitions under the Application object type.

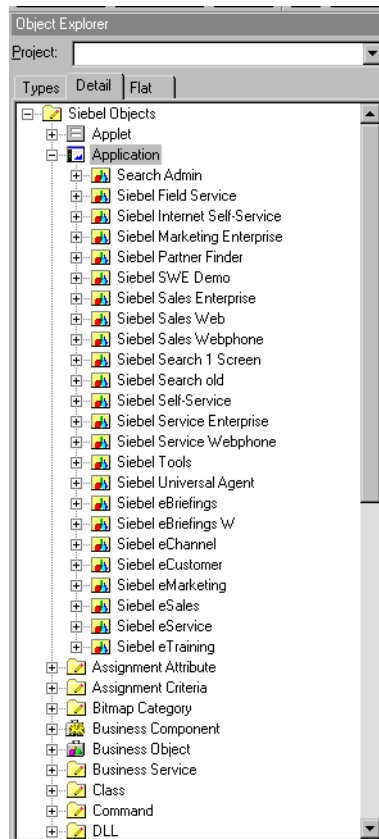


Figure 18. Object Explorer: Detail Tab

Flat Tab

The Flat tab (shown in [Figure 19](#)) of the Object Explorer shows all object types (parent and child) in a single, alphabetically-arranged list, without displaying the parent-child relationship.

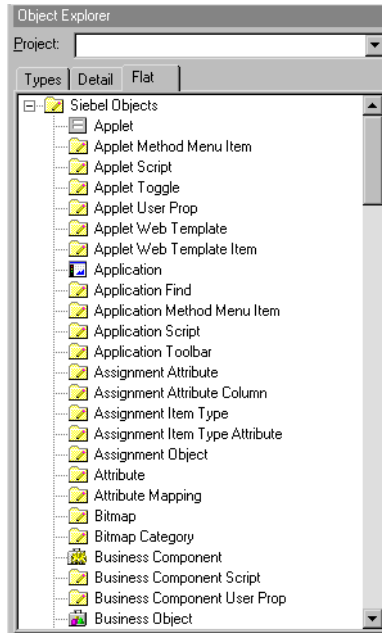


Figure 19. Object Explorer: Flat Tab

The Flat tab view helps you:

- Find a child object with an unknown parent.

For example, if you created a new field but do not remember what business component it is in, you can select the Field object type in the Flat tab and search the Name property for your field name. Each returned record has a parent property that provides the business component name.

- See how object definitions and properties are typically used.

Showing and Hiding Objects in the Object Explorer

Siebel Tools allows you to show or hide objects in the Object Explorer, allowing you to tailor your workspace to suit your needs.

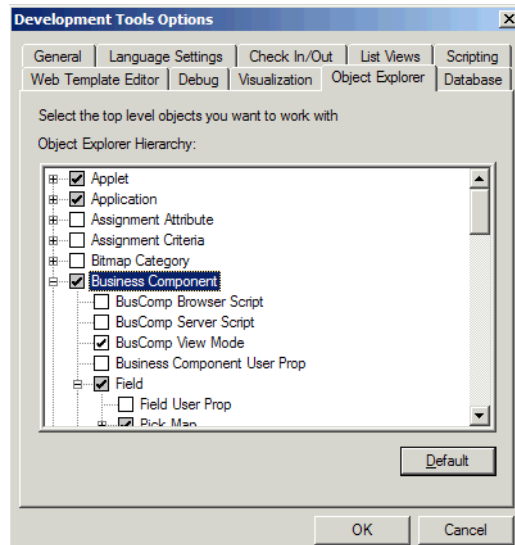
To show or hide objects in the Object Explorer

- 1 Choose View > Options.

The Development Tools Options dialog box appears.

- 2 Select the Object Explorer tab.

The Object Explorer Hierarchy is displayed.



- 3 Do one of the following:

- Show objects by placing a check mark next to the object name.
- Hide objects by removing the check mark next to the object name.

- Restore default settings by clicking the Default button.

When you select a top-level object such as Applet, all child objects are automatically selected. To hide child objects, you need to expand the parent object and remove the check marks from any child objects that you want to hide. The parent check box will then appear shaded to indicate that it contains child objects that are not selected to show.

- 4 Click OK.

The objects you selected appear in the Object Explorer Window.

Filtering Object Types by Project

Use the Project field at the top of the Object Explorer to filter objects by project. For example, you can set the Project filter so that only the object types associated with the Account project appear in the Object Explorer, as shown in [Figure 20](#).

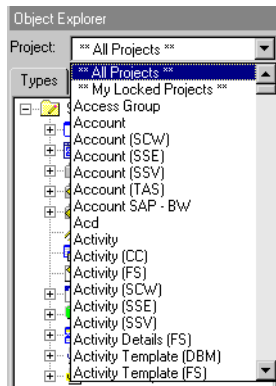


Figure 20. Selecting Projects from the Projects Box

Object List Editor Window

The Object List Editor window displays the object definitions for the object type currently selected in the Object Explorer. If the object selected in the Object Explorer window is a second- or third-level object, two Object List Editor windows are displayed—the object definition for the type selected in the Explorer is in the bottom window. In the example given in [Figure 21](#), the top-level object is Applet, the specific applet is Activity Assoc, and the available Web templates are Base (selected) and Query.

In the same figure, the pencil icon (to the left of the applet name) indicates that the applet has been locked by the Siebel Tools user, so that modifications to it can be saved.

The next applet in the list, Activity Assoc Applet - My, is printed in red, indicating that this object is *inactive*. An *inactive* object is one with deactivated database mappings (see [“EIM Interface Tables”](#) on page 228).

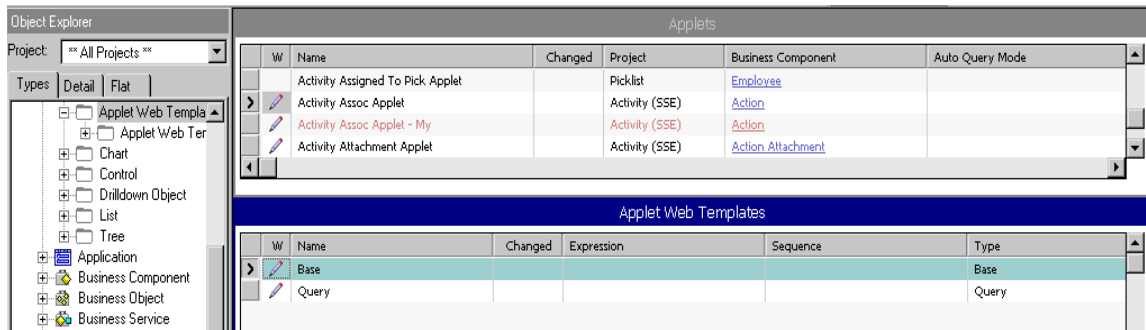


Figure 21. Object List Editor—Top-Level and Second-Level Objects

Other Windows

In addition to Object Explorer and Object List Editor windows, other windows can be made visible on your Siebel Tools screen. These can be accessed from View > Windows. The available choices are Properties Window, Applets Window, Controls Window, Bookmarks Window, or Web Templates Window. The windows are shown in [Figure 22](#) through [Figure 27 on page 112](#).

Properties Window. The Properties window (shown in [Figure 22](#)) displays the property settings for the object definition currently highlighted in the Object List Editor.

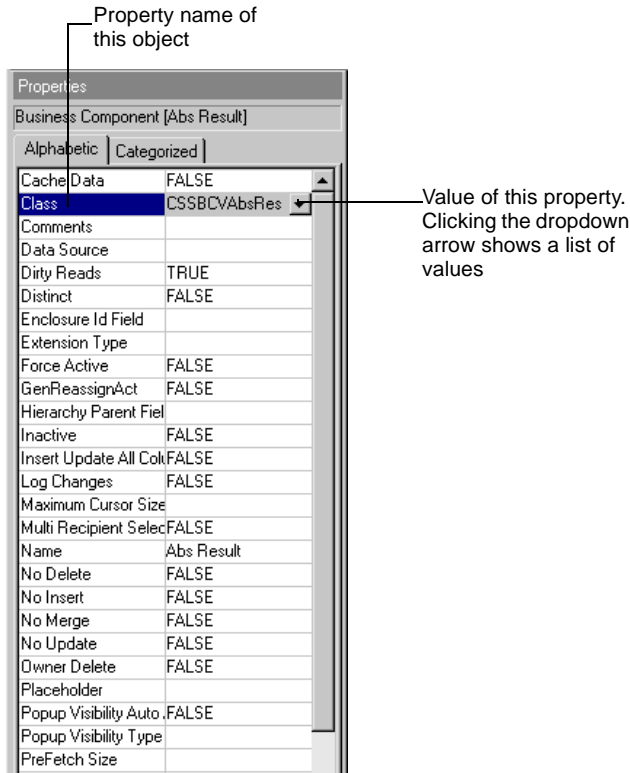


Figure 22. Properties Window

Applets Window. The Applets window (shown in Figure 23) displays the applets that are part of the business object selected from the Business Object list field in the Applets window. Clicking one of the applets opens it in the Web View Editor. You can also drag and drop the applet icon into the view layout editor.

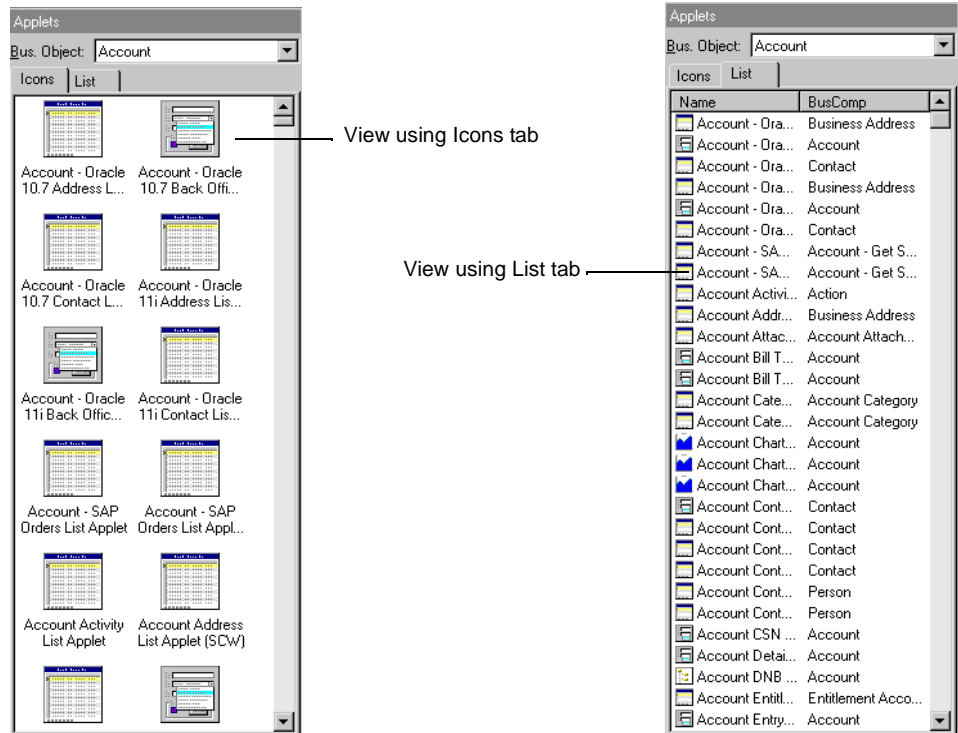


Figure 23. Applets Window—Icons Tab and List Tab

Controls/Columns Window. The Controls/Columns window (shown in Figure 24) displays controls or columns in an applet layout editor available for configuration when editing an applet layout in the Web Applet Editor. You drag and drop the control or column icon into the placeholder in the Web Applet Editor.

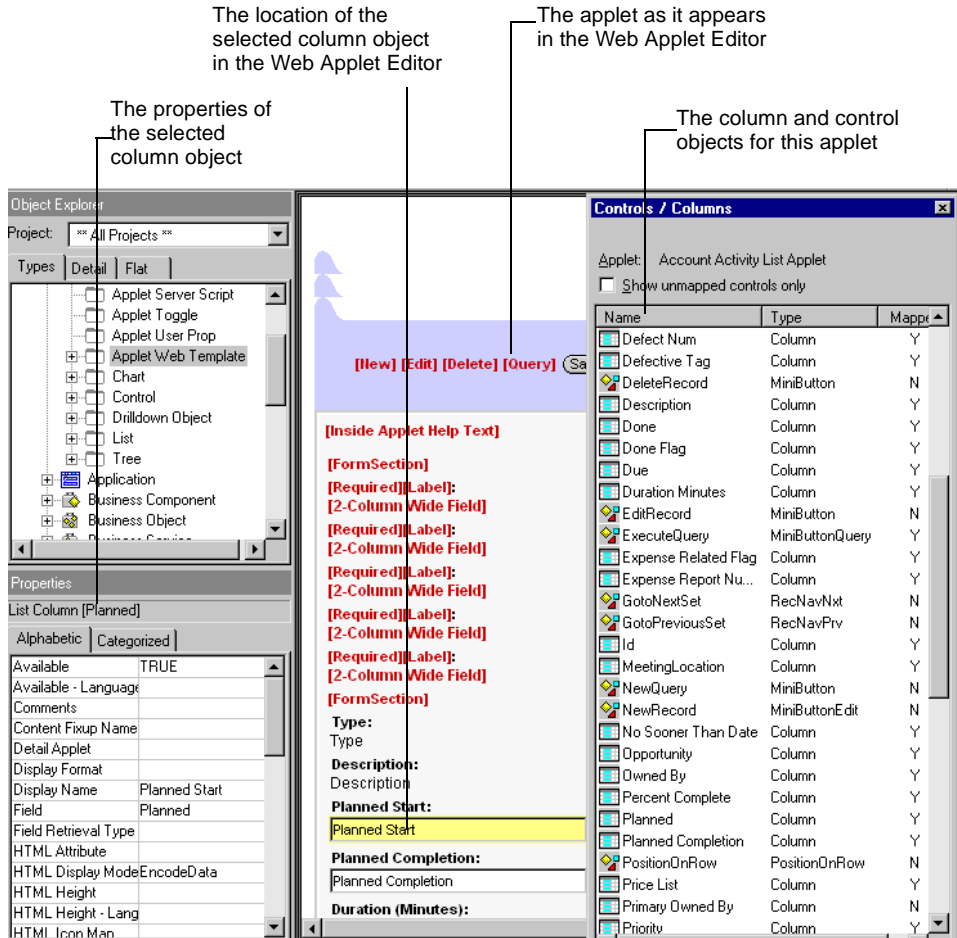


Figure 24. Controls/Columns Window

Web Templates Window. The Web Templates window (shown in [Figure 25](#)) shows or hides the Web Template Explorer window, a Windows Explorer-like listing of Web templates. Clicking on an item in the Web Template Explorer displays the HTML source code of the Siebel Web Template (.swt) file for review or editing in the main window. The template files are shown as a parent-child window. The Web Template combo box in the Web Template window allows you the choice of displaying all Web Templates in the Explorer, the top level Web Templates only, or individual Web Templates. A template file can be edited by right-clicking in the HTML code window for that template.

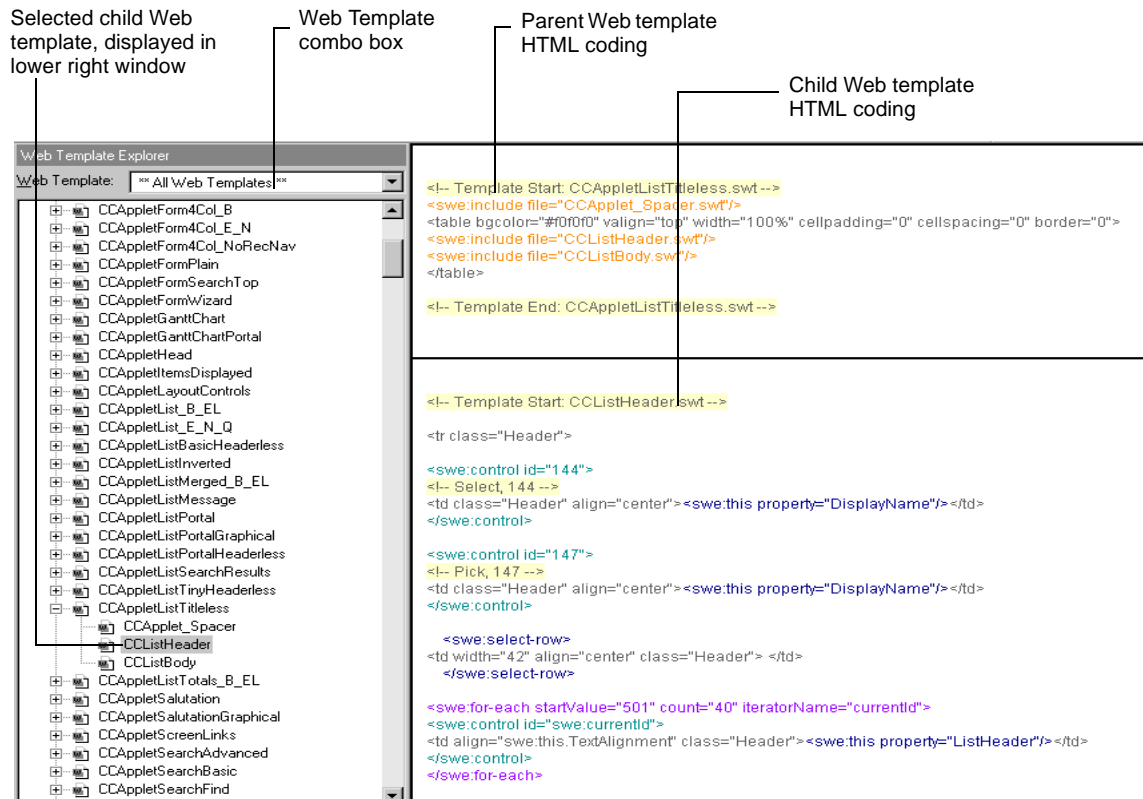


Figure 25. Web Template Explorer

Bookmarks Window. The Bookmarks window (shown in Figure 27) allows you to add shortcuts to frequently used objects in Siebel Tools. Open the Bookmarks window by clicking the Bookmark List icon in the toolbar (Figure 26). Add a bookmark by clicking the Add Bookmark icon.

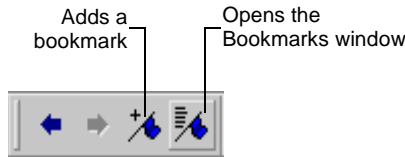


Figure 26. Bookmark Toolbar and Icons

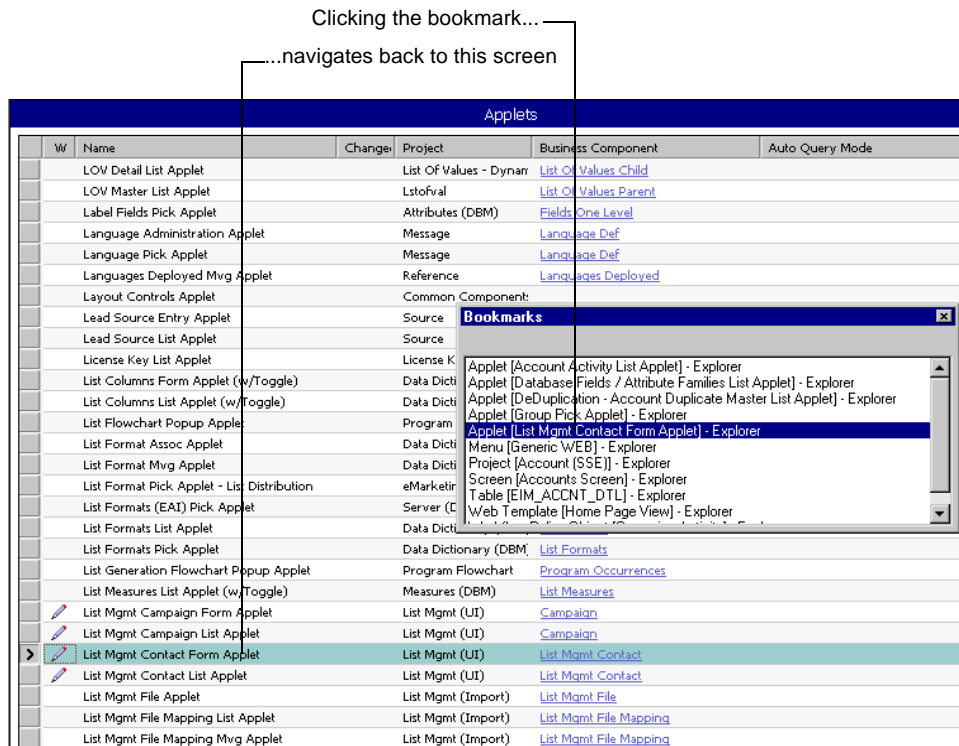


Figure 27. Bookmarks Window

Hiding the Windows

You can control whether or not the Object Explorer or the Properties, Applets, Controls, Web Template, or Bookmarks windows are visible.

To hide the windows

Do one of the following:

- For the Object Explorer, choose View > Object Explorer to remove the check mark indicating that the window is visible. Alternatively, press CTRL + E to hide the Object Explorer window.

For the Properties, Applets, Controls, Web Template, or Bookmarks windows, choose View > Windows > [Name of] Window to remove the check mark.

- For any of the windows, click the window to make it active, then click the right mouse button. Select Hide from the right-click menu.

Docking the Windows

You can let the Object Explorer or Properties, Applets, Controls, Web Template, or Bookmarks windows float, moving and sizing to fit your needs, or dock the window in a corner of the main window.

NOTE: You cannot dock or float the Object List Editor window. It is always docked by default, and you cannot float it.

To dock the windows

- Drag the window to the area of the main window where you want to dock.

To undock the windows

- Right-click the window and select Docked.

To prevent a window from docking when it is being moved

- Hold down the CTRL key during the move.

Image Preview

You can preview images in Tools, not only in BMP format, but also other common image formats, such as GIF, JPG and PNG.

To preview these images

- 1 In the Object Explorer, choose Bitmap Category > Bitmap.
- 2 Select a record in the Object List applet and right click.
- 3 Select Preview from the context menu.

You can navigate through multiple bitmap records and keep this window open.

Drilldown

In Siebel Tools, you can drill down from one object to another, when the second object is shown as the value of one of its properties. You can drill down on an object if the name of the object is underlined in blue (indicating a hyperlink). You can also drill back up, using the Go Back arrow in the History toolbar. You cannot drill down to object types if they are not displayed in the Object Explorer. (To display an object in the Object Explorer window, see [“To show or hide objects in the Object Explorer” on page 105.](#))

NOTE: Siebel Tools users must be assigned the Developer responsibility to use drilldowns. Users are assigned this responsibility in the Siebel employee application.

Viewing Object Definitions

You can view object definitions in the Object List Editor or the Properties window.

To view definitions in the Object List Editor

- 1 In the Object Explorer (Types tab), select the parent type object whose definition you want to view.
- 2 Expand the tree to view the child object type, if one exists.

3 View the object definition in the Object List Editor.

Figure 28 shows the field-level definition for the Contact business component, as displayed in the Object List Editor windows. The top applet shows the Buscomp Contact, and the bottom applet shows the fields.

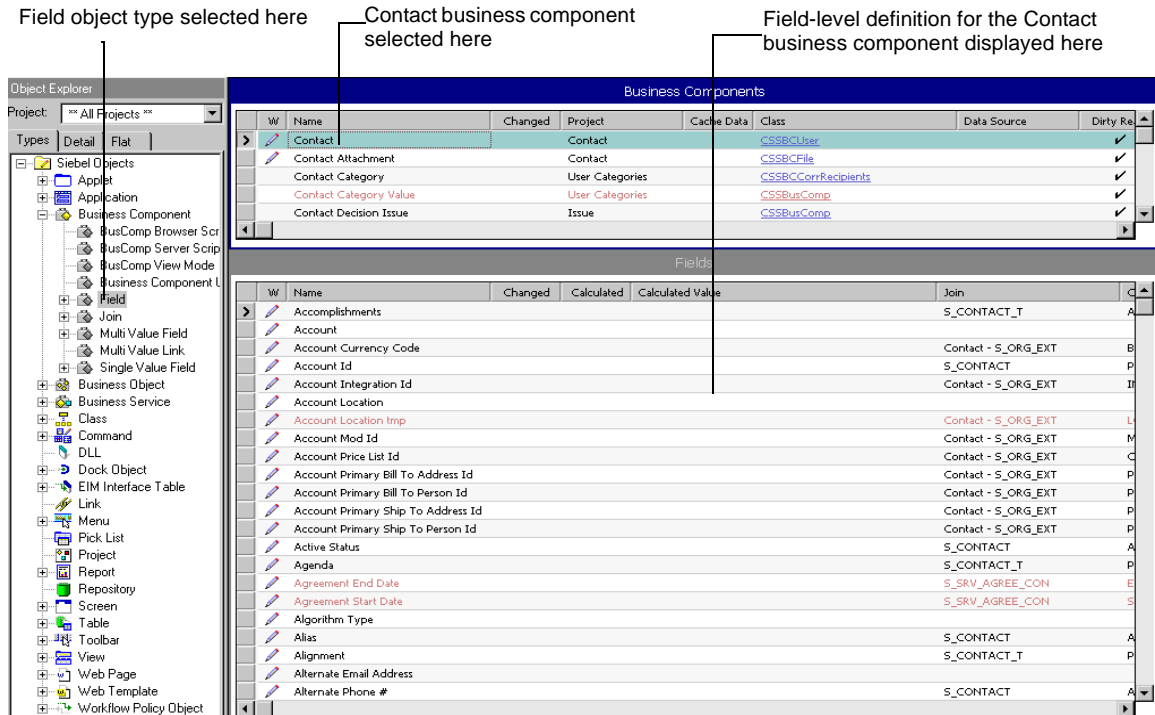


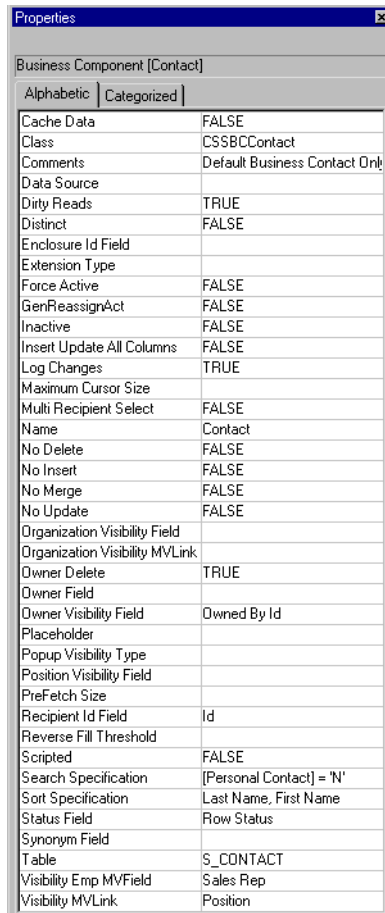
Figure 28. Viewing Field-Level Information in the Object List Editor Windows

The Properties window also displays (in alphabetical order) the definition of the selected object—the value is shown next to the property name.

By default, the Properties window is closed when you start Siebel Tools.

To open it so you can view an object definition, choose View > Windows > Properties Window.

Figure 29 shows the field-level definition for the Contact business component, as displayed in the Properties window.



Business Component (Contact)	
Alphabetic	Categorized
Cache Data	FALSE
Class	CSSBCCContact
Comments	Default Business Contact Onk
Data Source	
Dirty Reads	TRUE
Distinct	FALSE
Enclosure Id Field	
Extension Type	
Force Active	FALSE
GenReassignAct	FALSE
Inactive	FALSE
Insert Update All Columns	FALSE
Log Changes	TRUE
Maximum Cursor Size	
Multi Recipient Select	FALSE
Name	Contact
No Delete	FALSE
No Insert	FALSE
No Merge	FALSE
No Update	FALSE
Organization Visibility Field	
Organization Visibility MVLink	
Owner Delete	TRUE
Owner Field	
Owner Visibility Field	Owned By Id
Placeholder	
Popup Visibility Type	
Position Visibility Field	
Prefetch Size	
Recipient Id Field	Id
Reverse Fill Threshold	
Scripted	FALSE
Search Specification	[Personal Contact] = 'N'
Sort Specification	Last Name, First Name
Status Field	Row Status
Synonym Field	
Table	S_CONTACT
Visibility Emp MVField	Sales Rep
Visibility MVLink	Position

Figure 29. Viewing Field-Level Information in the Properties Window

NOTE: The Properties window does not display the Project and Changed properties.

Modifying, Copying, and Creating New Object Definitions

All the object definitions required for a standard Siebel application are already present when you install it. However, you can modify object definitions or create new ones in the course of application configuration.

- You modify object definitions in the Object List Editor or the Properties window.
- You copy and create new object definitions in the Object List Editor.

For more information about guidelines for modifying, copying, and creating new object definitions, see *Configuration Guidelines*.

- [“Object Definitions, Value Types, and Naming Conventions”](#)
- [“Modifying Object Definitions” on page 118](#)
- [“Creating a Copy of an Existing Object Definition” on page 120](#)
- [“Creating a New Object Definition” on page 121](#)
- [“Undoing New or Changed Object Definitions” on page 123](#)

Object Definitions, Value Types, and Naming Conventions

Object definitions can have the following value types:

- User-defined names
- Numerical values
- Boolean values (TRUE and FALSE)
- Siebel-defined constants
- References to the names of other object definitions

Example.

A field has the following values:

Name: Account Products

Text Length: 500

Read Only: FALSE

Type: DTYPE_TEXT

Column: PROD

As illustrated in the prior example, standard Siebel objects follow these naming conventions:

- Logical objects in the Business Object (for example, Account Products) and User Interface Objects layers use mixed case and embedded spaces.

It is recommended that you follow this convention in the Business Objects and User Interface Objects layers.

- Physical objects in the Data Object Layer (for example, DTYPE_TEXT and PROD) use uppercase and underscores.

This convention is enforced in the Data Object Layer.

NOTE: Always prefix the names of new objects with a meaningful text string. For example, if your company name is XYZ Industrial Products, Inc. you might prefix the name of new objects with XYZ. You will find it a valuable convention during both development and upgrade.

For more information about naming objects, see *Configuration Guidelines*.

Modifying Object Definitions

To modify an object definition in the Object List Editor

- 1** Be sure the project the object is a part of is locked.

For information about locking projects, see [Chapter 18, “Application Development Projects.”](#)

- 2** In the Object Explorer window, select the relevant object type.
- 3** In the Object List Editor window, select the object definition you want to change.

- 4 Use the TAB key to move the cursor to the specific value you want to change.

NOTE: It is recommended that you use the TAB key to move from property column to property column in the object definition—if you use the mouse you might unintentionally change the value of a Boolean property.

- 5 Type in a new value, or pick a value from the picklist (if one is provided).
- 6 To commit your changes, click anywhere outside the modified row (or move outside the row with the UP or DOWN arrow).

A check mark appears in the Changed column (for more information about the Changed column, see [“Understanding the Changed Flag and Pencil Icon” on page 135](#)).

To modify an object definition in the Properties window

- 1 Be sure the project the object is a part of is locked.

For information about locking projects, see [Chapter 18, “Application Development Projects.”](#)

- 2 In the Object Explorer window, select the object type.
- 3 In the Object List Editor window, select the relevant object definition.
- 4 Open the Properties window, if it is not already open (View > Windows > Properties Window).
- 5 Select the current value, and then type in a new one.
- 6 To commit your changes, click anywhere outside the modified row (or move outside the row with the UP or DOWN arrow).

A check mark appears in the Changed column (for more information about the Changed column, see [“Understanding the Changed Flag and Pencil Icon” on page 135](#)).

Creating a Copy of an Existing Object Definition

CAUTION: Objects that are copied are not automatically upgraded to a new Siebel product release. Modify existing object definitions whenever possible, rather than create new ones, in order to maximize the upgradability of your changes and minimize maintenance costs. Should it ever be necessary to make a copy of an Applet, Business Component, Report or Integration Object, set the Upgrade Ancestor property of the copied object to refer to the original object so that the copy will be upgraded appropriately. For more information on the Upgrade Ancestor property, see [Chapter 17, “Repositories.”](#)

To create a new object that is a copy of an existing object

- 1 Lock the project to which the object belongs, if it is not already locked.

NOTE: The project must be selected from among those that have been locked.

- 2 In the Object Explorer window, select the relevant object type.
- 3 In the Object List Editor, locate the object definition to copy, and click anywhere in the row to select it.
- 4 Choose Edit > Copy Record.

A new row appears above the copied row, containing identical property values. The Changed flag is checked (for more information about the Changed column, see [“Understanding the Changed Flag and Pencil Icon” on page 135](#)).
- 5 Enter a new value for the Name property.
- 6 Click in the Project field.
- 7 In the picklist that appears, select the name of a currently locked project to which to assign the new object.
- 8 If necessary, modify any other relevant properties and child objects.
- 9 To commit your changes, click anywhere outside the new row or move outside the row with the UP or DOWN arrow keys.

Creating a New Object Definition

Use the following procedure to create a new object definition.

To create a new object definition

- 1** Lock the project that will contain the object you intend to add.

For information about locking projects, see [“Locking Projects Directly” on page 1010](#).

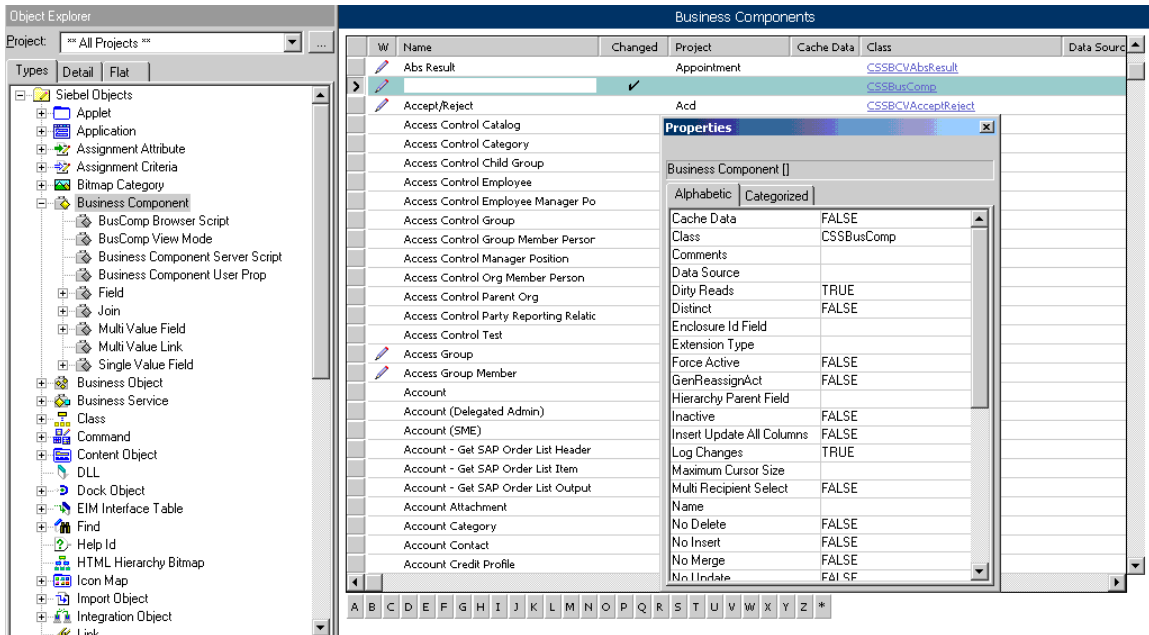
- 2** Select the relevant object type in the Object Explorer.

The Object List Editor opens, listing all object definitions of this object type.

- 3** Click in the Object List Editor window to make it active.

- 4 Choose Edit > New Record, or right-click and select New Record.

A new record appears. The following figure shows a new business component record.



- 5 Enter property values in the new row in the Object List Editor.

At a minimum, these consist of the object definition's Name property and Project property. Other properties may also be required, depending on the type of object definition you are creating. All required properties must be filled in order for the new object definition to be saved.

NOTE: Object names should not contain punctuation characters.

- 6 To commit your changes, click anywhere outside the new row or move outside of the row with the UP or DOWN arrow keys.

NOTE: New Object Wizards can also be used to create new object definitions.

Undoing New or Changed Object Definitions

The Edit > Undo Record menu option provides the means to discard your new record or modifications as long as you are still positioned on that record.

Once you reposition to a different record, the new record or changes are committed, and the Undo Record option is no longer available.

Validating Object Definitions

As you modify or create objects, it is very important to validate their definitions also. Validating object definitions is generally a good practice. It should be one of the first things that a developer does if a configuration changes produces a run-time error.

NOTE: The validation process is time consuming. However, you can continue working in Siebel Tools while the validation is run.

Validation is based on a set of rules that help make sure that your configuration changes are logically consistent with existing object definitions and one another. Validating a parent object validates all child objects as well.

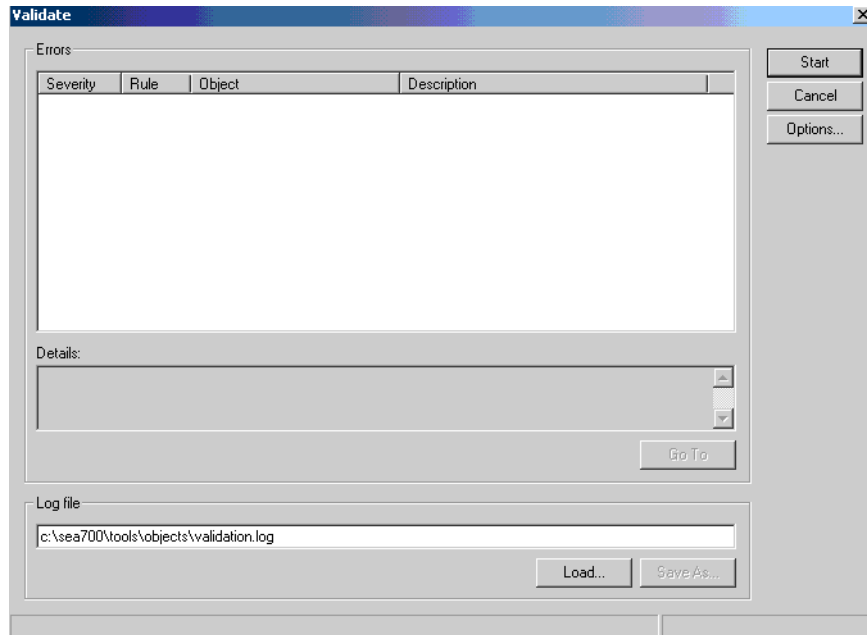
The rule that checks for invalid object references is the most important. An invalid object reference occurs when one object (an applet, for example) references another object (a business component) that has been inactivated or deleted.

To validate an object

- 1 Select the object or objects you want to validate.

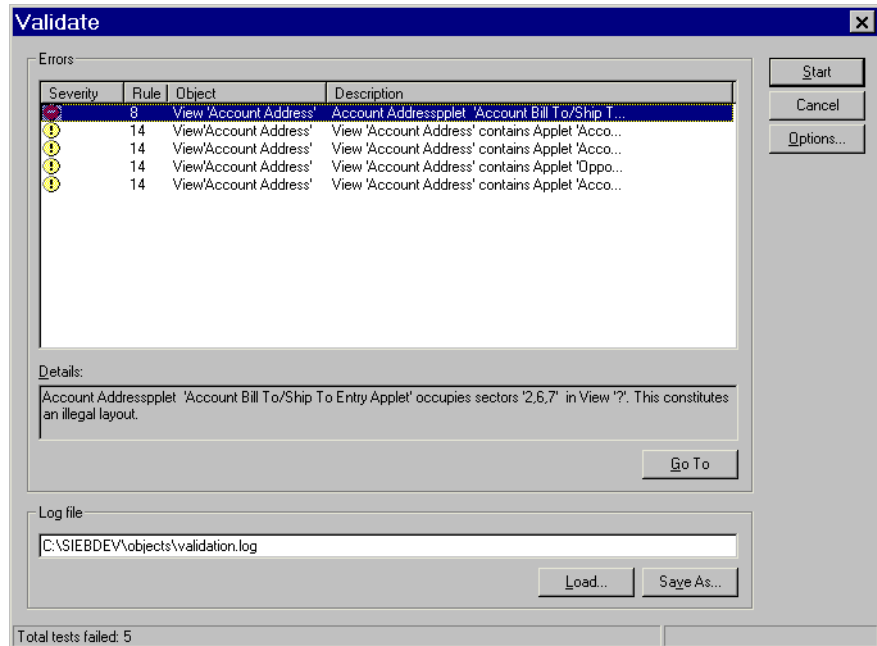
- 2 Right-click and select Validate, or choose Tools > Validate Object.

The Validate dialog box appears.



- In the Validate dialog box, click Start.

Violations of the rules currently being enforced appear in the Errors list, as shown in the following figure.



User Interface Elements in the Validate Window

- Errors list table.** Displays the results of the validation process. Each row in the list table identifies a rule violation for a specific object definition. You can double-click any of the errors to drill down to the specific object definition that contains the error. You can sort the rows by a particular list column by clicking the heading of that list column. You can also widen and narrow list columns by dragging the right or left border of the heading cell.

The Errors list table contains the following columns:

- **Severity.** An icon appears in this list column for each violation row. It indicates whether the violation is a warning (yellow icon with an exclamation mark) or an error (red icon with a minus sign). Errors cause the compiled application to generate run-time errors.
- **Rule.** An integer value appears in this list column, identifying the rule that has been violated. Rules are listed in order of rule number in the Validation Options window (shown in [Step 2 on page 127](#)).
- **Object.** The name of the object definition that failed validation.
- **Description.** The description of the error or warning. It is normally cut off by the right boundary of the list column. To view the complete text, click a validation row, and it appears in the Details text box.
- **Details text box.** The full text of the error or warning message for the currently selected row in the Errors list table.
- **Go To button.** Select an error message row and click Go To to navigate to the corresponding object definition in the Object List Editor. Alternatively, you can double-click the error message.
- **Log File text box.** Path and filename of a log file containing the list of validation errors and warnings. To save a list of validation rows as a log file, click Save As, navigate to the right directory, and specify a filename. You can then reload the list of error and warning validations at a later time by using the Load button, rather than by repeating the validation process.
- **Load button.** Opens a previously saved log file and displays its list of validations in the Errors list table.
- **Save As button.** Saves the current list of validation rows as a log file.

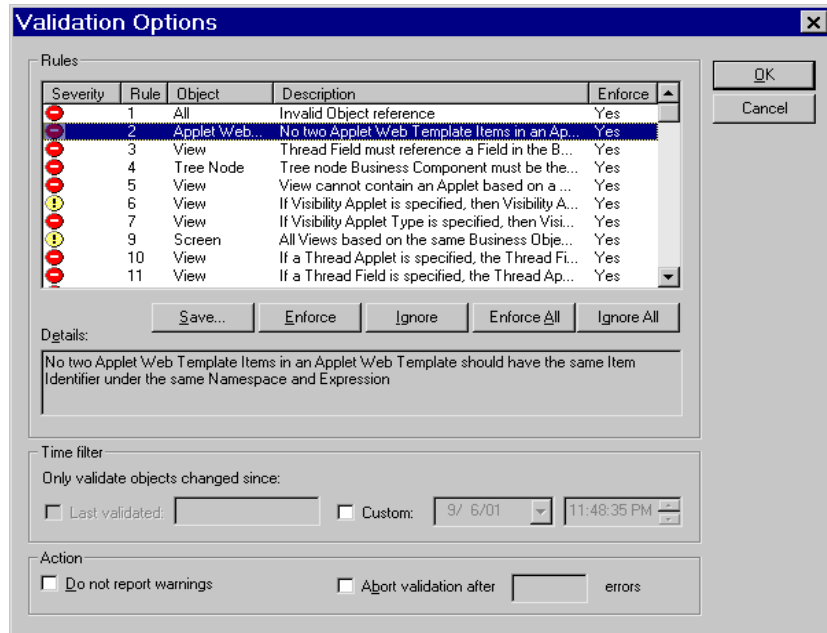
You can change whether or not certain validation options are enforced.

To change validation options

- 1** In the Validate window (see [Step 2 on page 124](#)), click Options.

- Change items in the Enforce column (shown in the following figure) from Yes to No, or vice versa, for rules you want to enforce or not enforce.

You do that by selecting a row and clicking the Enforce or Ignore button.



User Interface Elements in the Validation Options Window

- Rules list table.** Lists all rules that can be enforced during validation.

Each row in the list table identifies a rule for a specific object type (or All). You can sort the rows by a particular list column by clicking the heading of that list column. You can also widen and narrow list columns by dragging the right or left border of the heading cell.

The Rules list table contains the following list columns:

- Severity.** An icon appears in this list column for each rule row. It indicates whether the rule generates a warning (yellow icon with an exclamation point) or an error (red icon with a minus sign).

- **Rule.** The integer value that identifies this rule.
- **Object.** Either the single object type that this rule applies to, or All.
- **Description.** The description of the rule. It is normally cut off by the right boundary of the list column. To view the complete text, click a rule row, and it appears in the Details text box.
- **Enforce.** A Yes or No value for each rule row. Yes validates all object definitions of the object type identified in the Object list column. Yes/No values in this list column are changed using the Enforce, Ignore, Enforce All, and Ignore All buttons.
- **Save button.** Saves the current set of rules and their state (enforced or ignored) to a text file you specify. Other settings are saved to the preferences file automatically when you press ENTER.
- **Enforce button.** Changes the Enforce list column value in the selected row from No to Yes.
- **Ignore button.** Changes the Enforce list column value in the selected row from Yes to No.
- **Enforce All button.** Changes all values in the Enforce list column to Yes.
- **Ignore All button.** Changes all values in the Enforce list column to No. This has the effect in the next validation of not validating any object definitions.
- **Time filter check boxes.** The repository Validator should be used only in conjunction with the Time Filter, to avoid validating objects that are not being used. Choose one of two time filters:
 - **Last validated check box.** When checked, validates only objects changed since the date you enter into the corresponding date box.
 - **Custom check box.** When checked, validates only objects changed within the date range you enter into the corresponding date boxes.
- **Details text box.** The full text of the rule description for the currently selected row in the Rules list table.
- **Action check boxes.** Choose refinements in Validator actions.

- **Do not report warnings check box.** When checked, reports errors only, not warnings. It also changes the Enforced setting of all warning rules to No.
- **Abort validation after check box and text box.** When the check box is checked and an integer value appears in the text box, the system stops validating after the specified number of errors is reached. By default, the validation process continues to run until it is completed or cancelled.
- **OK button.** Saves the validation option settings and closes the Validation Options dialog box.
- **Cancel button.** Discards the Validation Options settings and closes the Validation Options dialog box.

Compiling and Testing Object Definitions

After you have modified object definitions, you need to compile the changes to a repository file (.srf). The .srf file is updated with the new object definitions and they become available in any instances of the Web client reading that repository file.

NOTE: An application's configuration file (CFG) includes a parameter (RepositoryFile) that defines the repository file (.srf) to read at run time.

You can compile projects or selected top-level object definitions. Compiling projects is more efficient when you have many changes in one or more projects. Compiling object definitions is more efficient when changes are isolated to only a few object definitions.

Compiling Projects

The Object Compiler allows you to compile all projects or a subset of all object projects. To compile a subset of projects, you must have compiled all projects at least once.

CAUTION: Avoid compiling a subset of projects into an .srf file, unless the .srf file was built from a full compilation from the same database. In particular, you should avoid doing compiling to the generic .srf file included in standard Siebel eBusiness Applications.

When you compile a subset of projects, the Object Compiler will not remove inactive top-level objects from the repository file, but it will remove inactive child objects. For example, if you inactivate the Name list column in the Account List Applet, and then compile the Account SSE project, the Name list column will be removed from the repository file. However, if you inactivate the Account List Applet, and then compile the Account SSE project, the Account List Applet will not be removed.

To compile projects

- 1 Choose Tools > Compile Projects.

The Object Compiler appears with the list of projects displayed.

- 2 Select the projects you want to compile.
- 3 In the Siebel repository file field, select Browse and choose the appropriate .srf file.

Typically you will compile to the .srf file used by the local instance of the Web client that you are using to test. The path to this .srf file is specified in the application's CFG file.

CAUTION: Do not attempt to compile to or modify the default .srf file used by Siebel Tools that is displayed in the Object Compiler window—usually in C:\sea7xx\tools\OBJECTS\siebel.srf. This file is locked because the Siebel Tools program itself reads the .srf file as it runs. If you attempt to compile to this filename and path, you will receive an error message.

- 4 Select the Auto-start web client option if you want to automatically start a local instance of the Web client when the compile process finishes.

You need to have specified the location of the Siebel executable, the application configuration file, and other relevant settings in the Development Tools Options Dialog box. For information on how to do this, see [“Setting Up Debug Options” on page 133](#).

- 5 Click Compile.

The object definitions in your repository are compiled to the .srf file you specified. The changes are immediately available in any instances of the Web client that are reading the .srf file. See [“Testing Repository Changes” on page 132](#).

Compiling Single Objects or a Group of Objects

You can compile a single object or a group of top-level objects of the same type. For example, if you modify the UI for several applets, rather than compiling entire projects, you can compile only the applets that have changed.

To compile single objects or a group of objects

- 1 Select an object or group of objects of a particular top-level type (for example, applet).
- 2 Right-click and select Compile from the menu.

The Object Compiler dialog box appears with a list of selected objects displayed.

- 3 In the Siebel repository file, select Browse and choose the appropriate .srf file.
- 4 Click Compile.

The object definitions are compiled to the .srf file you specified. The changes are immediately available in any instances of the Web client that are reading the .srf file. See [“Testing Repository Changes” on page 132](#) for more information.

Compiling the Siebel Repository Using the Command-Line Interface

You can also compile projects using the command-line interface. The command-line interface is invoked from the `siebdev` executable using the command switch `/bc`. This command switch performs a full compile. For multilingual deployments you can also set the Tools active language for the compile. The `siebdev.exe` is located in the `Bin` directory of the Siebel Tools installation directory.

The syntax for the `/bc` switch is:

- `siebdev.exe /c <config file> /d <data source> /u <user name> /p <password> /tl <language> /bc <Siebel Repository> <SRF file>`

For example, the following command compiles the Siebel Repository into `siebel.srf` with the active language set to Japanese.

- `siebdev.exe /c tools.cfg /d sample /u sadmin /p sadmin /tl JPN /bc "Siebel Repository" siebel.srf`

If no file path is specified for the `.srf` file, the file will be compiled into the `objects` directory under the Tools installation directory. Otherwise, it will be compiled into the specified directory.

NOTE: The Auto-start web client option that is available in the Object Compiler dialog box is not available when compiling using the command-line interface.

Testing Repository Changes

Siebel Tools allows you to test changes to the repository immediately after you compile them. After compiling, local instances of the Web client that are running and are reading the repository file to which you compiled are immediately updated with the changes. For more information about compiling, see [“Compiling and Testing Object Definitions” on page 129](#).

When compiling object definitions and testing the results locally, consider the following:

- You must have a local instance of the Siebel Web client on your machine to test the repository changes locally. See the *Siebel Web Client Administration Guide* for more information on installing a local instance of the Web Client.

- If a local instance of the Web client is installed but it is not open, you can select an option in the Object Compiler to automatically open a local Web client and read the most current repository. See [“Compiling Projects” on page 130](#) for more information.
- For repository changes to appear in local instances of the Web client, the Web client must be reading the repository (.srf) file to which you compiled. For example, if two local Web clients are open and they are both reading the .srf file you just compiled, repository changes will appear in both Web clients. However, if two local Web clients are open and one is reading .srf file A and the other is reading .srf file B, then repository changes to .srf file A will only appear in the Web client reading .srf file A.
- If you cancel the compile process while the Web client is running, the Web client goes into hibernate mode and remains in this state until you either successfully recompile or exit the Web client process. If you successfully recompile, the Web client will then open and read the new repository file. If you want to exit the Web client, right-click the Web client icon in the taskbar and then choose Exit.

Setting Up Debug Options

The debug options defined in the Development Tools Options dialog box provide the run-time settings for opening an instance of the Siebel Web client in the following situations:

- When the Auto-start web client option is selected in the object compiler. See [“Compiling and Testing Object Definitions” on page 129](#) for more information.
- When starting an instance of the Web client by selecting Debug > Start. You typically use this option when debugging Siebel eScript or Siebel VB. See *Siebel eScript Language Reference* and *Siebel VB Language Reference* for more information.

The settings defined the Debug tab of the Development Tools Options dialog are stored in a user preference file that is named `login_ID&application_name.spf` and located in `tools_install\BIN`.

To setup Tools to automatically open the Siebel Web client

- 1 In Siebel Tools, choose View > Options.

The Development Tools Options dialog box opens.

- 2 Click the Debug tab and then complete the following information:

Field	Example Value	Description
Executable	Siebel.exe	Name of the executable that is opened in debug mode or automatically opened after the compile process.
CFG file	D:\sea7xx\client\BIN\ENU\uagent.cfg	Name and location of the configuration file for the application.
Browser	C:\Program Files\Internet Explorer\iexplore.exe	By default, the mobile Web client will use IE 5.x, but users can specify a different browser.
Working Directory	D:\sea7xx\client\BIN	The directory that contains the Siebel executable.
Arguments	/h	Opens the watch window that allows you to trace the application.
User name	SADMIN	User name used to log into the test application.
Password	SADMIN	Password to log in to the test application.
Datasource	Sample	Local database to which the local Web client connects.

- 3 Click OK.

Understanding the Changed Flag and Pencil Icon

After you edit a record, a check mark appears in the Changed field of the object definition. This indicates that changes have been made to the contents of the corresponding record since a particular date and time. (This date and time is set using the General tab of the View > Options menu.) Lack of a check mark indicates that the object definition has not been changed since the date and time specified in Tools > Options.

The pencil icon in the first (W) column of an object definition indicates that the object is locked and editable. Figure 30 shows a changed field (Access Control Category) and many locked objects (pencil icon in column before the Name column).

Name	Changed	Inactive	Locked	Locked By Name	Locked Date	Language Locked	Comments
Access Group							
Account	✓		✎	SADMIN	9/7/01 3:36:10 PM	ENU	
Account (SCW)	✓		✎	SADMIN	9/8/01 11:34:47 PM	ENU	
Account (SSE)	✓		✎	SADMIN	9/8/01 10:10:30 PM	ENU	
Account (SSV)							
Account (TAS)		✎					
Account SAP - BW							
Acid							
Activity							
Activity (CC)							Activity - Com
Activity (FS)							Field Engineer
Activity (SCW)							
Activity (SSE)							
Activity (SSV)							
Activity Details (FS)							Field Engineer

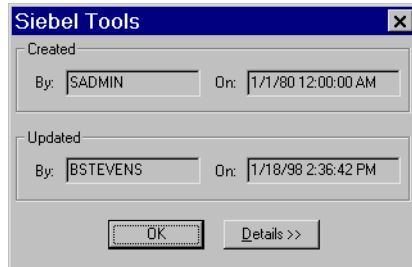
Figure 30. Changed Project (System) and Locked Objects (the Pencil Icon)

The Changed flag cascades upwards through its parents. That is, when an object definition is edited or created, the changed flag is set for its parent object definition, if any, and for the parent object definition of that parent, and likewise up through the hierarchy.

To determine by whom and when a record was created and last updated

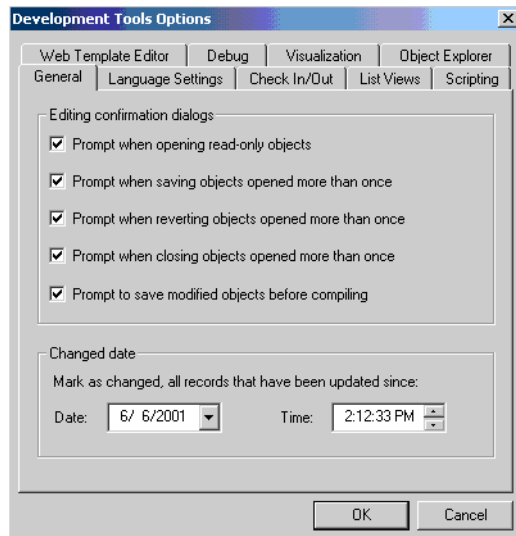
- 1 Select a record in the Object List Editor.
- 2 Choose Help > About Record from the menu bar.

- 3 The Siebel Tools dialog box appears (this dialog box displays the user, date, and time for record creation and update).



To update the Changed date manually

- 1 Choose Tools > Options.
- 2 Click the General tab.



- 3 Set a date and time.
- 4 Click OK to save the Changed date.

Using Queries to List Object Definitions

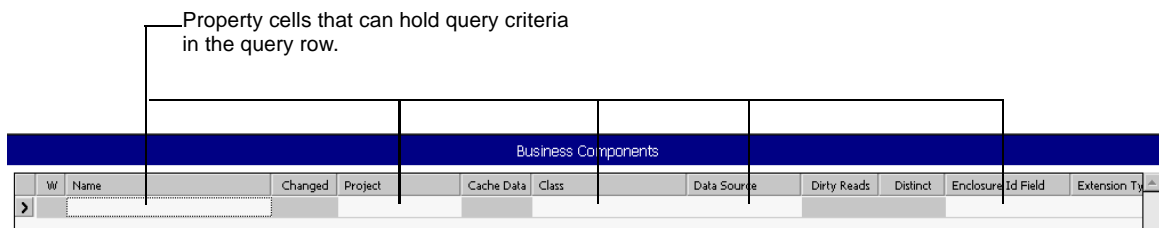
The Object List Editor provides a Query feature that uses a query-by-example (QBE) metaphor to let you narrow the list of object definitions in the current Object List Editor window. For example, when you are in the list of business components, you can use the Query feature to display only those business components in the Contacts field, or only those having names that start with Opportunity, or any other restrictions you can specify for particular properties.

An Object List Editor query is a search for object definitions based on matching values in one or more properties. The queries can be simple, one-condition queries or compound, multiple-condition queries. You can create, refine, and activate queries from the Query menu or from the List toolbar. (Refine means impose a further restriction on the current Object List Editor query by re-executing it with an additional constraint.)

To create and execute an Object List Editor query

- 1 Navigate to the Object List Editor window listing object definitions of the relevant object type.
- 2 Choose Query > New Query from the menu bar.

In the Object List Editor window, the list of object definitions is hidden, and a query row appears, as shown in the following figure.



- 3 In each of one or more property cells in the query row, enter a value for which the query will look for a match.

These values may be single literal values such as Opportunity List Applet, or they may include wildcard symbols. In TRUE/FALSE properties, a check mark represents TRUE.

- 4 Press ENTER or Choose Query > Execute Query from the menu bar.

The resulting list of object definitions in the Object List Editor consists of only those meeting the set of criteria you specified.

To restore the Object List Editor window to its prequery state

- 1 Choose Query > New Query from the menu bar.
- 2 Press ENTER or choose Query > Execute Query from the menu bar.

The list of object definitions in the Object List Editor is restored to its prequery state.

Simple Queries

A simple query finds information based on one condition. [Table 8](#) lists the operators you can use to create a simple query.

Table 8. Simple Operators

Operator	Description
=	Equal to
<	Less than
>	Greater than
< >	Not equal to
< =	Less than or equal to
> =	Greater than or equal to
*	Any number of characters (including none) may take the place of the asterisk (*)
?	Any one character matches the question mark (?)
IS NOT NULL	Searches for non-blank fields
IS NULL	Searches for blank fields
LIKE	Searches for values starting with the indicated string
NOT LIKE	Searches for values not starting with the indicated string

Table 8. Simple Operators

Operator	Description
“ ”	Searches for strings that contain special characters, such as a comma (,)
EXISTS ()	Searches for values in a multi-value group
[~]	Forces the case of the text string to whatever follows the tilde

For more information on search specifications and operators and on Siebel data types, see *Siebel Developer's Reference*.

Compound Queries

Compound queries enable you to find information based on two or more conditions. There are three ways to create compound queries:

- Enter conditions in two or more property columns to find records that meet all the conditions. In other words, Siebel applications automatically connect these conditions with the operator AND. This method is the easiest way to create a compound query.
- Enter a compound query within a property field using the operators OR, AND, and NOT to create two or more conditions for that property.
- Enter a compound query using more than one field and compound operators AND, OR, and NOT. You can enter this type of query in any field. You might find it convenient to use the Description or Comments field, because it is typically the longest on a given screen.

When you create a compound query, follow the same basic steps you use to create a simple query.

Use parentheses to control the order in which a compound search is conducted. Expressions inside parentheses are searched for first (as they appear left to right). [Table 9](#) lists the unique operators for compound queries. Use these operators *in addition to* the operators you use to create a simple query.

Table 9. Compound Operators

Operator	Description
AND	All the conditions connected by ANDs must be true for a search to retrieve a record.
OR	At least one of the conditions connected by the OR must be true for a search to retrieve a record.
NOT	The condition modified by this operator must be false for a search to retrieve a record.

For more information about compound operators, see *Siebel Developer's Reference*.

Searching the Repository for Object Definitions

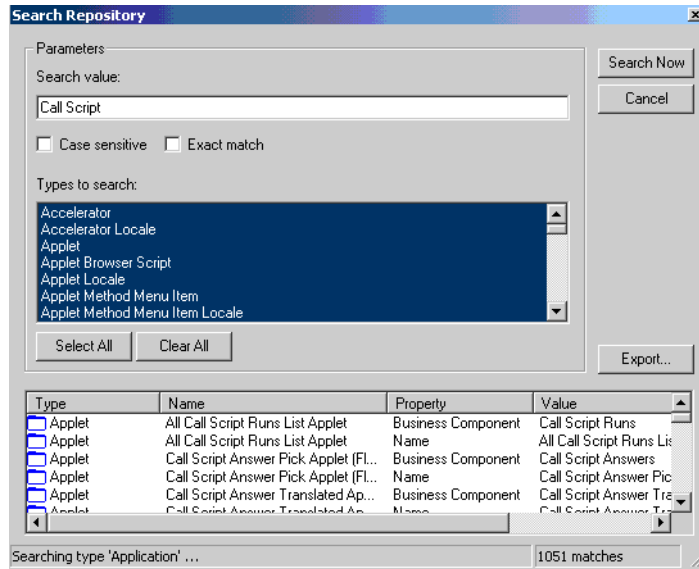
You can use the repository search facility to search for object definitions based on text in their names (or other properties) and their object types. It is a quick way to locate one or more object definitions when you know all or part of their names or some other property.

NOTE: This is a time-consuming task.

To display the Find In Repository window

- Choose Tools > Search Repository.

The Search Repository dialog box appears.



User interface elements in the Find in Repository window:

- **Search Value text box.** Enter the search string text to be located in the names and property values of object definitions.
- **Case Sensitive check box.** Check if you want only those object definitions whose property values contain the search string with the same capitalization. Leave empty if matching capitalization is unnecessary.
- **Exact Match check box.** Check if you want only those object definitions whose property values exactly match the entire search string.

- **Types to Search list box.** Object type or types to search for. By default, all object types in this list are selected. You can choose a single object type to search by selecting it. CTRL-clicking and SHIFT-clicking can be used to select multiple object types. For better performance, search only the object type or types you need.
- **Select All button.** Selects all the object types in the Types to Search list.
- **Clear All button.** Deselects all the object types in the Types to Search list.
- **(Result Object Definitions) list table.** Lists all the object definitions found in the search. Double-click on an item in this list to bring it up in the Object List Editor. Double-clicking on an item in the Result Objects List has the same effect as doing an Object List Editor query that searches on the name of the object definition.

To restore the Object List Editor window to list all object definitions of the type selected in the Object Explorer, do a query with the asterisk (*) symbol in the Name column of the Object List Editor.

- **Search Now button.** Executes the search and lists the results in the (Result Object Definitions) list table.
- **Cancel button.** Stops the search process if a search is executing. Closes the Search Repository dialog box in Repository window.

The Result Object Definitions list has these columns:

- **Type.** Object type of the object definition returned by the search.
- **Name.** Name of the object definition returned by the search.
- **Property.** Name of the property of the object definition in which the search value was found.
- **Value.** Value of the property of the object definition in which the search value was found.

Getting Reports About Object Relationships

You can get reports about the relationships between certain object types in the repository. This section provides an introduction to using the reports facility; for more information, see *Siebel Reports Guide*.

The list of records that displays in a repository report is not dependent on the currently selected object definition in the Object List Editor—for example, if you select the Contact business component in the Object List Editor and generate the business components and fields report, the report will show *all* business components, not just the Contact business component.

To restrict a report to a single parent object definition—that is, one business component, business object, dock object, table, or workflow object—do a query in the Object List Editor (for the parent object type) that restricts the list to the one object definition you want to see.

The following example shows how to get the Tables report for S_ORG_EXT.

To get a Tables report for the S_ORG_EXT table

- 1** In the Object Explorer, select the Table object type.
- 2** Activate the Object List Editor for tables by clicking on it.
- 3** Choose Query > New Query from the menu bar.
- 4** Enter a value of S_ORG_EXT in the Name property and press Enter.
- 5** Choose Reports > Tables.

The generated report will provide information only for the S_ORG_EXT table.

You can use a similar approach to restrict a report to a range of object definitions that have a property value in common. The report will include only those object definitions satisfying the current query. For example, you can get a Tables report of all extension tables, a business components report of all business components of a specific class, or any of the reports restricted to a single field.

The current object type in the Object Explorer determines the list of reports in the Reports menu. Repository reports are listed by current object type in [Table 10](#).

Table 10. Reports Available for Each Object Type

Object Type	Report	Description
Applet	Applets by BusComp	Lists the applets in the repository alphabetically within each business component.
Business component	Business Component and Fields	Lists the fields in each business component alphabetically. For each field, the base column and join table, if any, are identified.
Business object	Business Object and Components	Lists the business object components in each business object. For each business object component, the business component and link are identified.
Dock object	Repository Dock Objects	For each dock object displays selected properties, and lists the member tables, visibility rules, and related dock objects.
Project	Project List	Lists all projects, and identifies the locking status, person locked by, and locked date for each.
Table	Tables	For each table displays selected properties, and lists the columns. The name, physical type, length, scale, comments, and various other properties are identified for each column.
Workflow Policy Object	Workflow Policy Objects	Lists the workflow components in each workflow object, and within each workflow component lists the columns.

Viewing Object Relationships: Visualization Views

You can use the Siebel Tools Visualization views to see how object definitions relate to one another.

To invoke the Visualization views

- Use one of two methods:
 - Choose View > Visualize > View Details, View Relationships, View Descendents, or View Web Hierarchy.
 - Right-click an object definition of the relevant object type in the Object List Editor, and choose the Visualization view you want. Not all of the Visualization views are listed for all objects.

The Visualization views are:

- **View Details.** Applicable to business components and business objects. The diagram displays how the business component map to underlying tables directly or through joins, and map to other business components through links.
- **View Relationships.** Applicable to business components and tables. For business components, the diagram displays how the business component links to other business components using multi-value link object definitions. For tables, the diagram displays how the table joins to other tables by way of join object definitions.
- **View Web Hierarchy.** Applicable to applets, applications, business components, screens, and views. The diagram displays the parent-child relationships between the selected object definition and its parent and child object definitions, as well as the parents of the parent object definitions and children of the child object definitions, up and down the hierarchy.
- **View Descendents.** Shows all objects which have the current object marked as their Upgrade Ancestor.

Details Visualization Views

The Details Visualization View for a business component displays how it maps to underlying tables using joins or to business components by means of multi-value links. The Business Object version of the diagram displays links between pairs of business components in the business object.

Business Component Version

The selected business component appears in a box at the left of the diagram. The business component on the left may be associated with one or more of these boxes on the right:

- **Table box for the base table.** The base table for the business component appears to the right of the business component without any intervening join or mvlink boxes between it and the business component.

The base table in [Figure 31 on page 147](#) is Table S_ORG_PRDEXT.

- **Table boxes for joined tables.** Tables that are joined to the business component using join object definitions have a join box between the business component and the joined table. The join box provides the name of the join.

The joined tables in [Figure 31 on page 147](#) are Table S_ORG_EXT and Table S_PROD_EXT.

- **BusComp boxes for linked business components.** Business components that are linked to the selected business component using a multi-value link have an mvlink box between the selected business component and the linked business component.

[Figure 31 on page 147](#) does not show any linked business components.

For the tables and business components displayed on the right, only columns (or fields) in use by the business component are listed, unless you click on the plus symbol icon labeled All columns or All fields in the corresponding table or BusComp box.

If you click the name of a field in the left-side BusComp box, an arrow appears from that field name to the corresponding column in the base table, joined table, or linked business component on the right. This is illustrated in [Figure 31 on page 147](#), where Vendor Location points to LOC.

Business Object Version

The business object version of the Details Visualization view for Admin Product Definition is shown in [Figure 31](#).

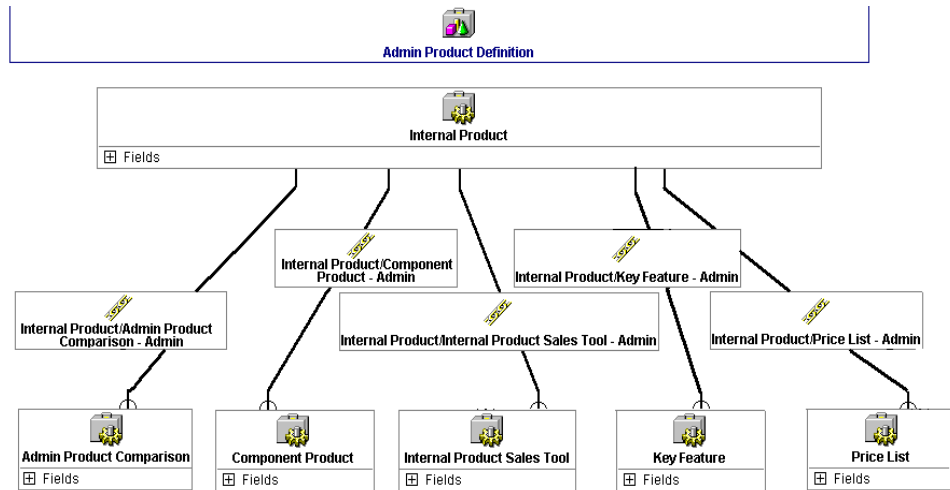


Figure 31. Business Object Version of the Details Visualization View

The selected business object is represented as a horizontal bar at the top.

All immediately subordinate business components in the selected business object appear in a row beneath the BusObj bar. In [Figure 31](#), Internal Product is the only business component immediately subordinate to Admin Product.

For each business component on the second row, all business components to which the business component has links appear on the fourth row.

On the third row (the row is staggered in [Figure 31](#)) are boxes representing the links between the two.

Plus and minus symbol icons identify lists that can be expanded or collapsed in a box. Business components expand to a list of fields, Tables expand to a list of columns, and Applets expand to a list of controls. [Figure 31 on page 147](#) illustrates this for three business components in a diagram; the two top ones have an expanded fields list, and the bottom one has a collapsed fields list.

You can use the up and down arrow keys to move up or down one level at a time in a visualization diagram. Use the left and right arrow keys to move between objects on the same level.

Right-click anywhere in a visualization diagram to display a shortcut menu that provides options pertaining to the entire diagram. These options are:

- **Expand All.** Expands all fields lists, column lists, and control lists in boxes in the diagram.
- **Contract All.** Collapses all field, column, and control lists.
- **Zoom.** Provides options for zooming in or out on the diagram.
- **Style.** Provides three options governing the general style of the diagram and the boxes in it. The default is Outline. 3D Border is similar, but it is on a gray background and shows beveled edges on the boxes. Iconic provides an icon in each box, which helps distinguish the object types visually.
- **Edit Definition.** Opens the Object List Editor with the currently selected object definition in the diagram displayed in the List Editor.
- **Edit Layout.** When a view or applet object definition is selected in a visualization diagram, the View Designer or Applet Designer is opened with that object definition displayed.

If the Properties window is displayed at the same time as the window containing a visualization diagram, the object definition whose properties appear changes as you select different object definitions in the diagram. Use this feature to navigate from object definition to related object definition to View properties. However, the properties cannot be edited when the Properties window is accessed this way.

To generate a Details Visualization View for a business component

- 1** Right-click a business component.
- 2** Choose View Details from the shortcut menu.

Relationships Visualization View

The Relationships Visualization View is available for business components and tables. For business components, the diagram displays links among business components using multi-value links. For tables, the diagram displays how the table relates to other tables using foreign keys.

Business Components Version

The business components version of the Relationships Visualization View appears in [Figure 32](#).

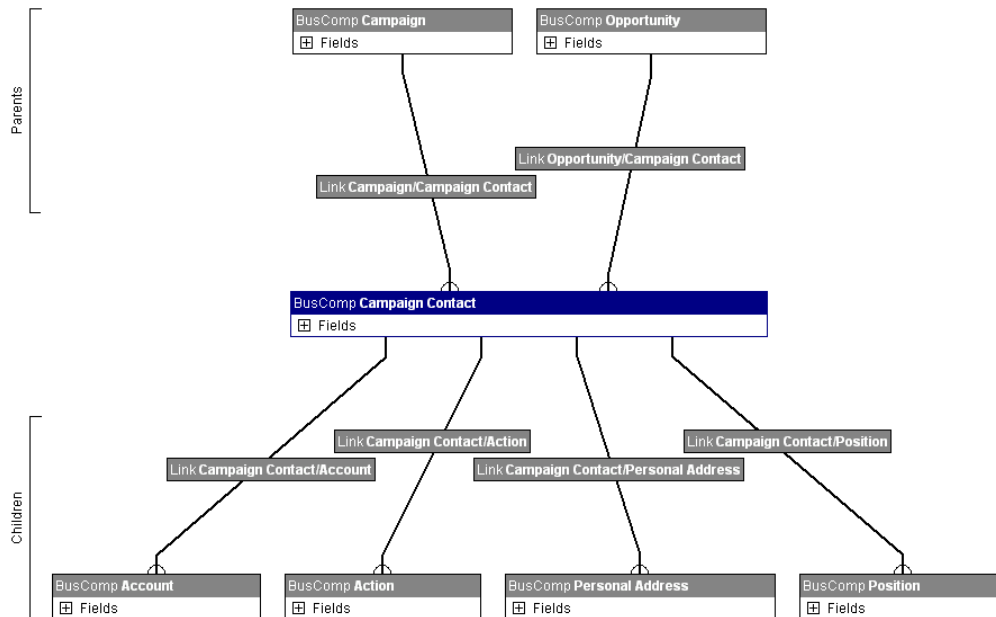


Figure 32. Campaign Contact Business Component—Relationships Visualization View

The selected business component (Campaign Contact in [Figure 32](#)) appears as a box in the center of the diagram. Business components that link to the selected business component appear in a row above it (connected by means of a row of link boxes). Business components that the selected business component links to appear in a row below it, again connected by a row of link boxes. The link boxes provide the names of the links.

To navigate within the business components version of View Relationships

- Double-click on a business component other than the initially selected one—it becomes the focus of the diagram.

To display a View Details diagram for a selected business component

- 1** Right-click a business component.
- 2** Select View Details from the menu.

Tables Version

The Tables version of the Relationships Visualization View appears in [Figure 33](#).

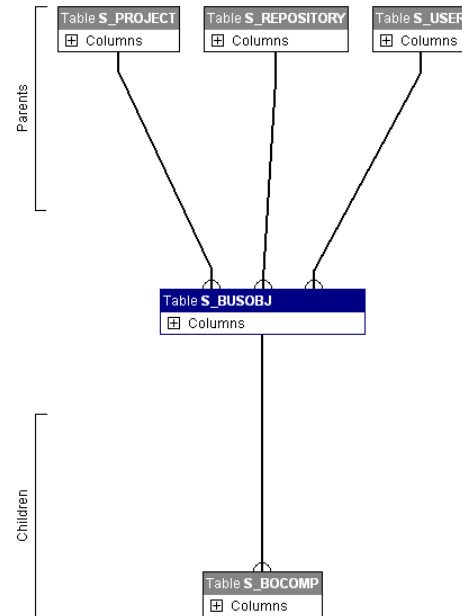


Figure 33. Table [S_BUSOBJ]—Relationships Visualization View

The selected table appears as a box centered in the diagram. The diagram shows the table's immediate foreign key relationships to other tables. Tables that have foreign keys to the selected table appear above it, and tables to which the selected table has foreign keys appear below it.

To navigate within the tables version of View Relationships

- Double-click on a table other than the initially selected one—it becomes the focus of the diagram.

Descendents Visualization View

The Descendents Visualization view (Figure 34) displays a list of all descendents of the selected object. Descendents all have their Upgrade Inheritance property set to the selected object and will be upgraded the same way. The Descendents Visualization view provides a convenient means to view Ancestor-Descendent relationships between top level objects. Users can select a descendent from the list for comparison and selective synchronization with the Ancestor or with each other, thereby simplifying the process of maintaining multiple copies of similar objects. For further information on Upgrade Inheritance, see Chapter 17, “Repositories.”

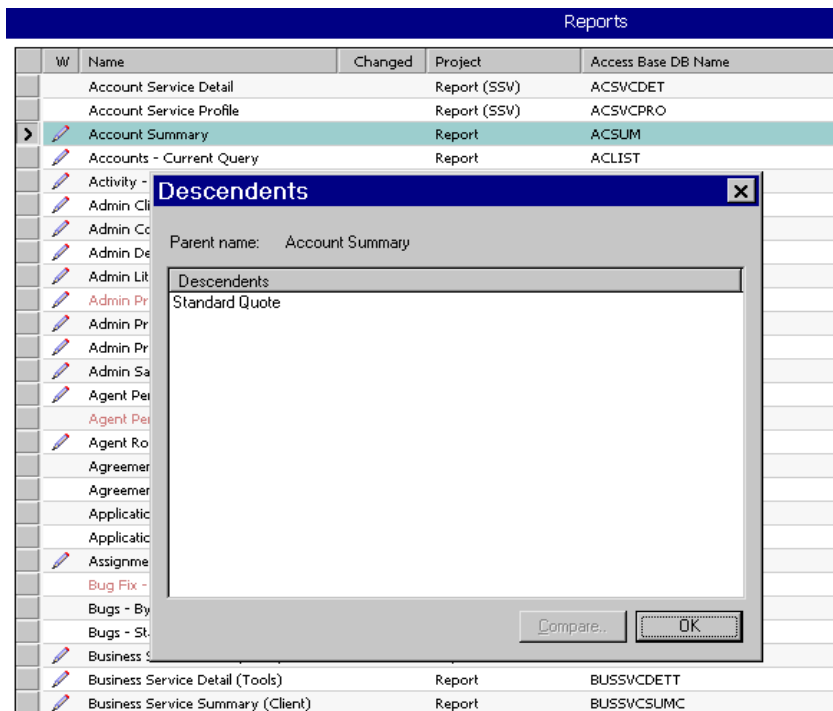


Figure 34. Descendents Visualization View

Web Hierarchy Visualization Views

The Web Hierarchy Visualization View is available for applet, application, business component, screen, and view object types. The diagram displays the parent-child relationships between the selected object definition and its parent and child object definitions, as well as the parents of the parent object definitions and children of the child object definitions, up and down the hierarchy.

A simplified example of the Web Hierarchy Visualization View appears in [Figure 35](#).

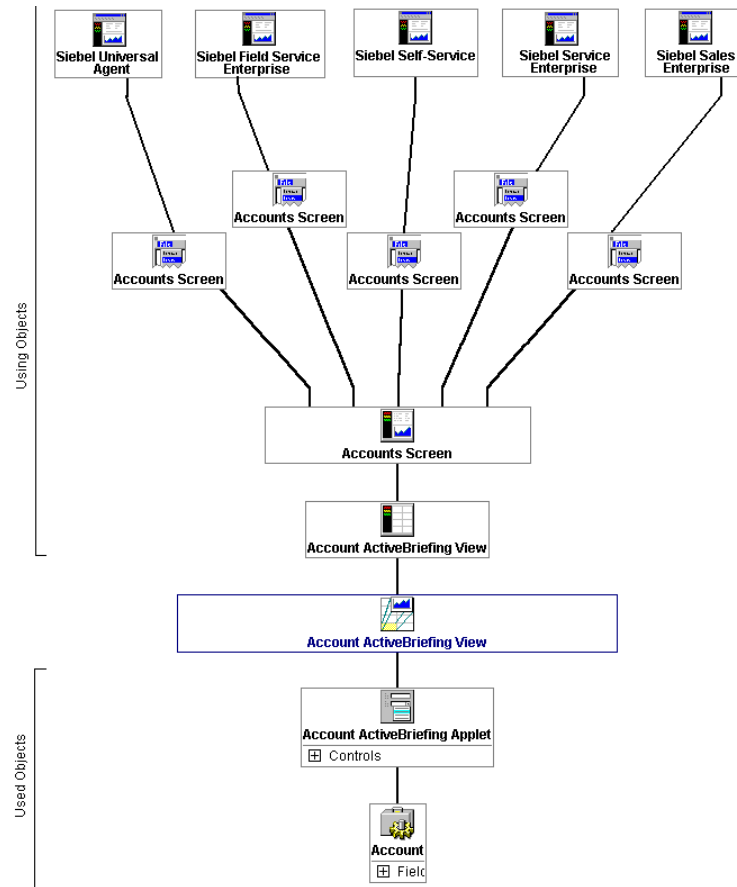


Figure 35. Web Hierarchy Visualization View

The selected object definition appears as a box in the center of the diagram. For this object definition, this diagram shows its parent-child relationships to other object definitions of various object types. Parent object definitions appear above the selected object definitions, and child object definitions appear below. The hierarchy starts with the parent application at the top level and includes screens, views, applets, and business components.

Because the entire hierarchy appears for an application, it is better to create View Hierarchy diagrams for object definitions lower in the hierarchy than the application level.

To navigate within a View Hierarchy diagram

- Double-click on an object definition other than the initially selected one—it becomes the focus of the diagram.

You can also navigate to the Details Visualization View for a business component in the Hierarchy diagram by right-clicking the business component and selecting the View Details option.

Siebel Tools Product Components

These features of Siebel Tools allow you to extend your application's functionality.

- [“Siebel Object Interfaces”](#)
- [“Siebel Database Extension Designer”](#)
- [“Siebel Application Upgrader” on page 155](#)
- [“Siebel Upgrade Inheritance” on page 156](#)
- [“Siebel Object Comparison and Synchronization” on page 156](#)

Siebel Object Interfaces

Siebel Object Interfaces allow you to access to the object definitions and data in Siebel applications by external programs through COM, CORBA, and Java Data Bean interfaces.

Siebel Object Interfaces provide open interfaces into the Siebel applications, supporting integration between Siebel applications—Siebel Sales, Siebel Marketing, Siebel Service, Siebel Field Service—and external applications.

For more information on the Siebel Object Interfaces, see *Siebel Object Interfaces Reference*.

Siebel Database Extension Designer

For developers who require extensions beyond built-in database extensions, Siebel Database Extension Designer provides a point-and-click interface for extending Siebel application tables. The Database Extension Designer allows new columns and new one-to-one tables to be added.

Advanced Database Extensibility, on the other hand, allows new stand-alone tables, one-to-many tables, and intersection tables to be added. Advanced Database Extensibility also allows interface table mappings and dock objects to be created from a wizard.

You can use these database extensions to capture data from new fields in application screens or from external sources using the Siebel Enterprise Integration Manager (EIM).

For more information on extending the database, refer to *Siebel Enterprise Integration Manager Administration Guide*.

Siebel Application Upgrader

The Siebel Application Upgrader reduces the time and cost of version upgrades by enabling you to acquire new features from the latest release while preserving the custom configuration changes made to the current repository. It notifies system administrators about conflicts between object customization and new releases, automatically merges differences between object definitions, and enables you to manually override and apply any changes. For more information on upgrading your application, see the upgrade guide for the operating system you are using and [Chapter 17, “Repositories.”](#)

Figure 36 shows the Siebel Application Upgrader.

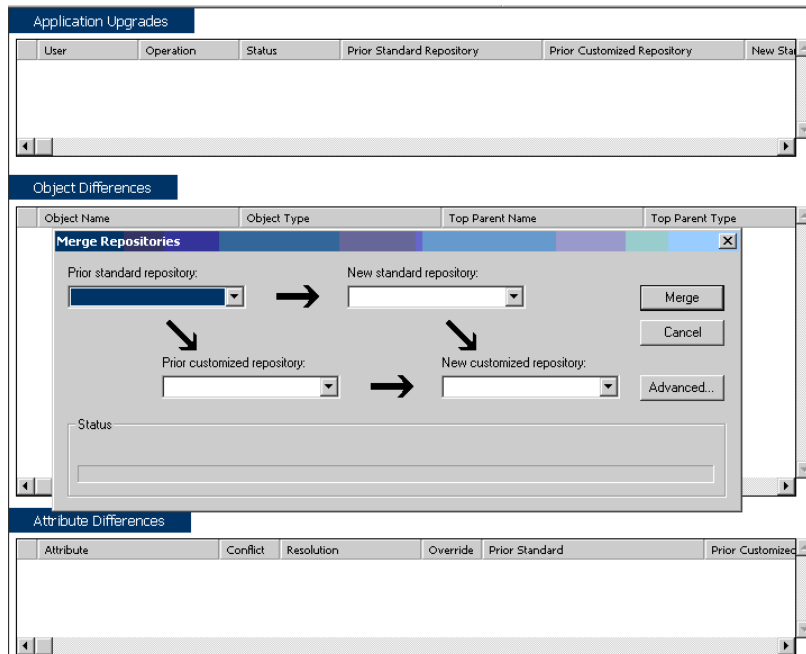


Figure 36. Siebel Application Upgrader

Siebel Upgrade Inheritance

Copied objects inherit some of the behavior of their ancestors, which makes it easier to upgrade Siebel applications. A new property called Upgrade Target allows copied objects to be upgraded in the same way as the ancestor objects from which they were copied. For further information, see [Chapter 17, “Repositories.”](#)

Siebel Object Comparison and Synchronization

You can view a side-by-side comparison of any two objects of the same type. You can select and copy properties and individual child objects from one object to the other.

Using this feature, you can propagate change made to an ancestor object to its descendants or other objects of a similar type. You can also compare properties of checked-out objects with their counterparts on the server. See [Chapter 17, “Repositories.”](#)

Siebel applications are delivered with a standard interface definition. Many Siebel customers use standard Siebel applications just as they are purchased; however, other customers configure the appearance and behavior of Siebel applications based on their organization's requirements. Siebel Tools is a development environment that allows you to configure Siebel applications. This chapter provides an introduction to configuring Siebel applications and an overview of the components and capabilities of Siebel Tools.

About Configuration

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

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

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

NOTE: Siebel VB and eScript are run on the server side; JavaScript is run on the client (browser) side.

This section helps you understand the many components of Siebel Tools that allow you to configure an application.

To see other methods of configuration in Siebel applications, see [“Other Ways to Customize Application Behavior”](#) on page 184.

Usage and Configuration of Non-Licensed Objects

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

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

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

Configuration Goals and Objectives

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

Key objectives for your configuration project should include:

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

If you follow this principle your configured application will be much easier to maintain and upgrade to future Siebel product releases. See the section on Upgrade Inheritance in [Chapter 17, “Repositories.”](#)

- Standardize configuration development.

For object naming guidelines, see [“Object Definitions, Value Types, and Naming Conventions”](#) in [Chapter 3, “Siebel Tools Fundamentals.”](#)

Follow these guidelines when you modify existing objects or create new ones.

- Achieve acceptable system performance.

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

- Build a consistent and intuitive user interface.

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

For information about configuring your Siebel application user interface, see [Chapter 3, “Siebel Tools Fundamentals.”](#)

Overview of the Web Configuration Process

Like other forms of software configuration, configuring a Web application is not a completely serial process. During some phases, it makes sense for multiple pieces to be configured concurrently. Furthermore, some tasks—most obviously testing and debugging—are iterative, more like a loop than a straight line. For this reason, feel free to modify the simplified, rather linear process recommended in this chapter to suit the needs of your team.

Configuring Siebel Web applications is a modular process that separates style and structure (style sheets and templates) from the binding (HTML display objects) to data. Style and structure are reusable across multiple HTML display objects, using Siebel templates. This means that modifications to the style and structure can be easily propagated to all HTML display objects.

Figure 37 depicts the relationships between style sheets, templates, HTML display objects such as applets and views, Business Object Components, and the final HTML output.

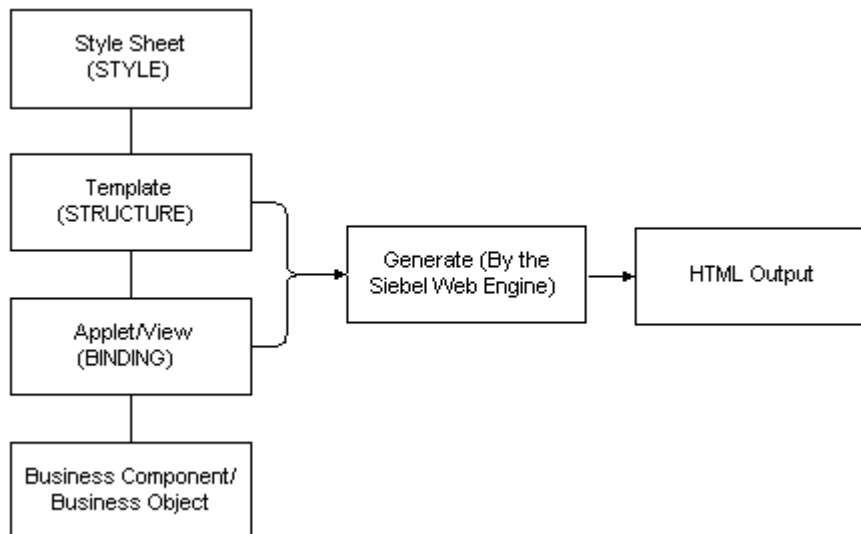


Figure 37. Relationships Between the Components in a Siebel Web Application

Using Siebel Tools and a text editor or HTML authoring tool, the Web application developer does the following:

- 1 Configures in Siebel Tools the business objects, applets, views, and all of the other normal elements of a Siebel application. Normally, you will be altering the definitions of objects in an existing application.

Identify the views, applets, and other parts of the Siebel Web application which you want to modify. Using Siebel Tools, configure HTML Display Objects (applets and views) that:

- Define new views and applets for your Web application
- Contain drilldowns to each other where appropriate, for example, from summary views to detail views

- 2** Associates the views and applets with Siebel Web Template (.SWT) files.
Use Siebel Tools to map the applets and views to their respective templates.
- 3** Modifies or create new .SWT files as necessary to integrate the appropriate corporate layout and formatting.
- 4** Establishes mapping between controls and list columns to corresponding placeholders in template files.
- 5** Compiles the repository changes into an .srf file.
- 6** Tests and debugs the application.
- 7** Deploys the application.
 - a** Copies the template files to the `webtemp1` directory in the Siebel Server installation.
 - b** Copies all new HTML and images to the public directory on the Web server machine.
 - c** Adds a link from the existing Internet or intranet site to the application.

Planning Considerations

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

- Assign a single developer or group the development role for a complete functional area.
 - For example, the group or individual person may develop a Web page and all the supporting logical business object definitions and data object definitions.
 - This approach typically enables different groups to implement in parallel.
- Assign a single developer or group to a specific architectural layer.
 - This approach takes advantage of the specialized expertise of developers—for example:

- The RDBMS specialists can implement extensions in the Data Objects Layer.
- The system architects can implement the Business Object Layer.
- The UI developers can implement the User Interface Objects Layer.
- Using a Web template requires each group to complete some work before another group begins.

Overview of the Application Development Process

This section describes a typical application development process.

- 1** Do a thorough business analysis of your organization's and users' needs, and get buy-in and time and resource commitments from the relevant organizations.
 - Can you meet the needs of your users with a standard Siebel application?
 - If not, what business needs will require changes to the application?
 - How can you assure success with your configured application?
- 2** Write a design document that includes:
 - The requirements that are being satisfied by the configured application
 - An ER diagram or text equivalent of the entity relationships
 - The names and descriptions of the business objects and business components required for your application, and how they relate to one another
 - Screen flow diagrams and a list of fields to be displayed on each applet
 - A description of your development environment and process, for example:
 - How the work will be divided up among participating developers
 - Naming conventions the development team will be required to use
 - How the application will be tested and rolled out to users

- The complete step-by-step procedures your development and test team will need to follow to complete the application
- 3** Have the design reviewed by:
 - Your participating organizations and users
 - The Siebel Expert Services organization
 - 4** Set up your application development environment—for example:
 - System and database environment
 - Developer workstations
 - 5** Develop the application:
 - a** Within Siebel Tools, create (or modify, if possible) the necessary object definitions:
 - Business components and business objects
 - User interface objects (for example, applets, views, and screens)
 - b** Modify your template files.
 - c** Compile your Siebel application and do unit testing.
 - 6** Using the tools available to you in the Siebel application environment (for example, Siebel Assignment Manager and Siebel Business Process Designer), implement the appropriate assignment and workflow rules.
 - 7** Extend the functionality of your application as needed through scripting using Siebel VB or Siebel eScript.
 - 8** Localize your application if the user interface is to be displayed in two or more languages.
 - 9** Do system and performance testing of your Siebel application.
 - 10** Iterate through the development steps until your design has been fully implemented and your application is running smoothly and meets your performance objectives.
 - 11** Introduce the application to your users and train them to use it effectively.

Siebel Object Definition Sequence

This section describes a typical sequence of steps for defining a Siebel application.

NOTE: This section focuses on the tasks listed in steps 4 and 5 of [“Overview of the Application Development Process”](#) on page 164.

Step One: Create Business Object

Create your business object as described in [Chapter 8, “Defining Business Objects and Business Components.”](#)

Step Two: Create Screens, Applets, and Views

In the first phase, you use Siebel Tools to configure business objects. The Business Model—business objects, business components, and so on—is defined using the current capabilities of Siebel Tools. Alternatively, you can use the business objects that are shipped with the Web application.

You configure the user interface by building and configuring HTML display objects—applets and views—based on the Business Object Model that you created in the preceding phase. This is where the applet and view Web layout changes occur.

At this point, you can begin to think about how your application will look and work in HTML. The HTML application is composed of several related display objects—page containers, views, and applets. Some of these objects are shared across application deployments, specifically applets and views.

For information about how to create objects, refer to [Chapter 9, “Logical User Interface Objects Layer.”](#)

Step Three: Associate Each Applet and View with the Correct Template

Now you will create an application definition in Siebel Tools. The Web application is assembled from the various HTML display objects and templates that were shipped with the base application, those that were customized in the configuration process, and Web pages that do not contain any Siebel elements.

The mapping between applets and views and their respective templates is achieved visually in Siebel Tools. [Figure 38](#) depicts how the Web application is assembled.

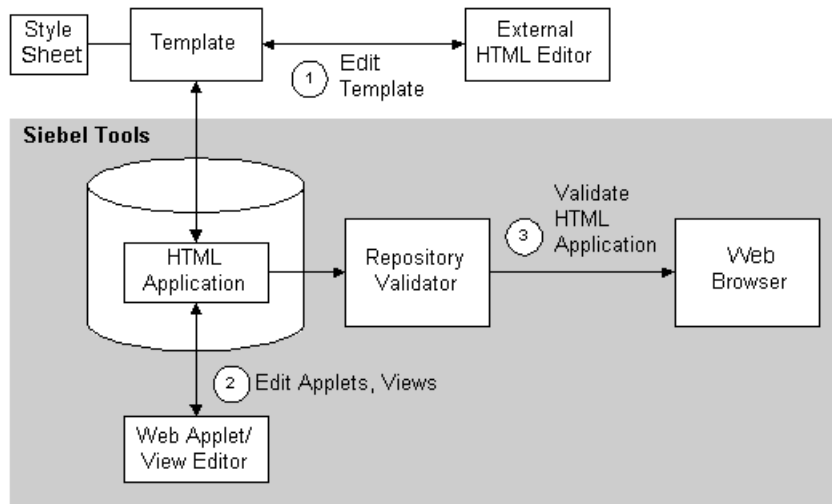


Figure 38. Assembling the Components into Your Web Application

For more information about this process, refer to [Chapter 10, “Logical User Interface Objects Configuration.”](#)

Step Four: Modify Templates as Needed to Create a Corporate Image

If you need to make changes to the structure in a template, you can open the template in an HTML editor and modify it. For example, if you are creating a list applet, you may want to make a column bold. Or you might need to make a global style change for the Web application to blend in with an existing Web site or support a branded look and feel to external users.

NOTE: The templates are located in the `webtempl` directory below the Siebel Server installation directory.

Step Five: Establish Mapping Between Controls and Templates

Templates are definitions of user interface layout and formatting. Each template contains placeholders for controls. In this phase, you create the mapping between each control in the repository definition of each applet and view with the placeholder in the corresponding template file. For more information about mappings between controls and templates, refer to “Mappings Between Controls and IDs” in [Chapter 14, “Physical User Interface Layer.”](#)

Step Six: Web Application Definition

The application can do nothing at this point because it lacks an application definition. Therefore, you must modify a version of the Web application in Siebel Tools and save the .SRF file for the modified Web application.

To create the Web application definition in Siebel Tools

- 1 Using the Object List Editor, create a new Application Object, entering the required attributes such as Name and Project.
- 2 Add a page container template to the Web application for your home page.
- 3 Select a default login page, error page, and acknowledgement page from the list of available Web pages.
- 4 Associate the Web application with the newly created screens.

The Siebel Web application model requires data to move from the database server and the Application Object Manager to the Siebel Web Engine and, ultimately, to the user's browser. When this data transfer needs to happen over low-bandwidth lines, as is often true in a Web client system, it becomes necessary to optimize the application in order to achieve the kind of performance users expect. The next section gives you some specific ways to modify the application definition for better performance.

Strategies to Optimize Application Performance

To modify your application to achieve better performance, do the following:

- **Reduce the number of objects.** Specifically, use fewer
 - Views per screen
 - Fields per appletOnly one page container is allowed.
- **Cut down on unnecessary duplication.** While the screen size available to present an HTML page does not limit the number of applets permitted per view, the need to shorten load time makes it a good idea to use multiple applets only when you consider it mandatory.
- **Minimize the number of multimedia objects such as graphics, audio, and video.**

Step Seven: Compile the Repository Changes into an .SRF File

In this phase, you should compile the .srf. For more information about this procedure, refer to [Chapter 17, "Repositories."](#)

When the application is run, the Siebel Web Engine does the following:

- 1 Reads the object definitions from the .SRF file.
- 2 Selects the specified templates.
- 3 Combines the object definitions and templates.
- 4 Retrieves necessary data from the underlying business objects and business components.
- 5 Presents the HTML output to the user.

Repository Validator

You can use the repository validator in Siebel Tools in conjunction with the Time Filter to detect errors in the configuration of the Web application and associated templates. This helps you detect invalid object references and unused objects, and verifies that all required attributes of controls and Web controls have been specified.

Step Eight: Test the Application

Having created default mappings between each object definition and an appropriate template, you can test the whole application.

Restart the Siebel Server, Web Server, and Gateway Server, link to the application's URL from a browser, and begin your testing. In addition to testing links, test the following:

- Logging in

NOTE: When coding an HTML link in an application that directly accesses a view or an eSmartsript, (for example, in a Web template or by script) you must make sure that the URL adheres to the case-sensitivity used to access the HTML application.

- Inserting new records
- Updating existing records
- Deleting records
- Performing queries
- Performing sorts
- Logging off

Testing with Browsers

The Siebel application templates have been tested with supported browsers. If you have not modified the Siebel default templates, there is no need to test the application in different browsers. However, if you have modified the Siebel default templates, you may want to test how the application behaves in different browsers.

Step Nine: Deploy the Application

When you have thoroughly tested the application on a development machine, you are ready to deploy it on your Web site.

Deploying the application involves the following major steps:

- 1** Copying the templates to the `webtemp1` directory in the Siebel Server installation
- 2** Copying all new HTML and images to the public directory on the Web Server machine
- 3** Adding a link from the existing Internet or intranet site to the application

For more information about installing the application and infrastructure, see *Siebel Server Administration Guide*.

Application Enhancement Through Scripting and Object Interfaces

The Siebel applications are primarily enhanced by creating and modifying object definitions. This provides many of the important benefits of the Siebel architecture, such as ease of application configuration, maintenance, and upgrade. Scripting is supported through three features in Siebel applications. These are Siebel Visual Basic (VB), Siebel eScript, and Browser JavaScript.

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

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

Server-Side Scripting

These are server-side Siebel VB or eScript scriptable object types, and respond to events on object manager objects (business components, applets, business services, and applications).

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

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

NOTE: Siebel VB is available on Windows platforms only.

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

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

- **Data validation routines.** These routines enforce specific business rules before or after performing record manipulation operations. Validation routines are performed before the user performs an update or an insert. The intent is to make sure that illogical or incomplete data is not entered into the database.
- **Data manipulation and computational routines.** These routines can be used to modify or analyze data.

- **Data transport routines.** These routines import and export small volumes of data between Siebel applications and other applications.
- **Application launching routines.** These routines launch external applications on the server side in response to Siebel events, and pass start-up parameters. Valid for browser scripting only.

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

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

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

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

Browser-Side Scripting

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

Browser scripts are written using Siebel Tools and can be associated with the following Siebel object types: applets, business components, business services, and applications. See [Figure 39 on page 175](#).

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

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

See *Siebel eScript Language Reference* for detailed information.

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

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

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

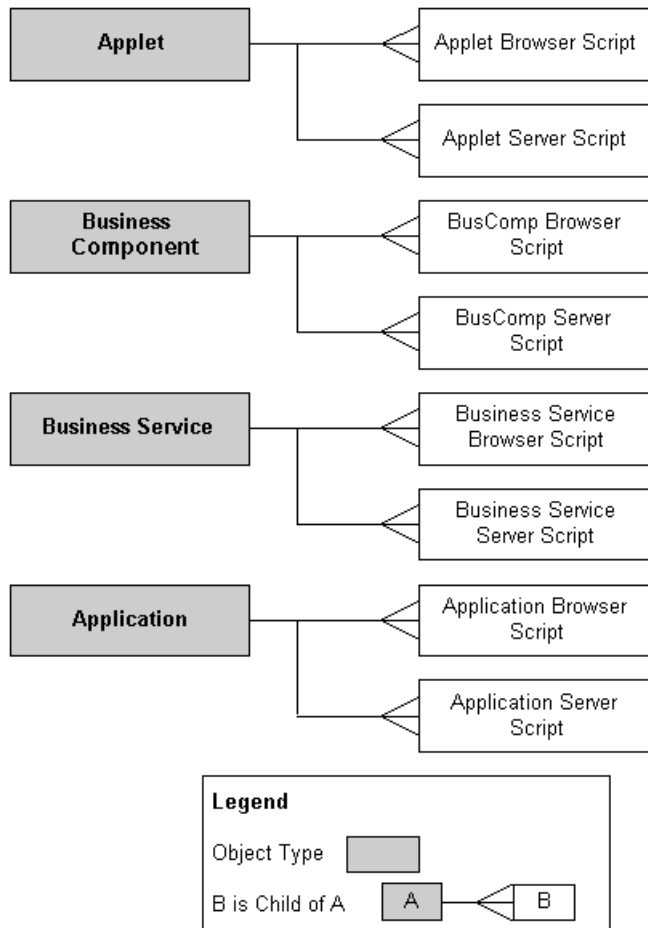


Figure 39. Customer Scripting Object Types

The following example, shown in [Figure 40](#), illustrates how you can configure client-side form field validation using Browser Script. It shows browser script for the Account BrowserBusComp PreSetFieldValue event handler.

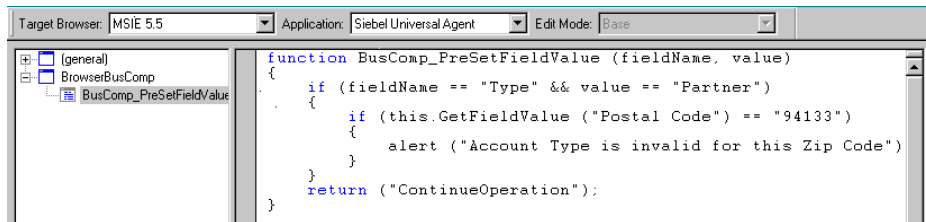


Figure 40. Example Browser Script

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

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

Generating Browser Scripts

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

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

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

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

```
tools_root\public\language_code\srf_timestamp\bscripts\all
```

- They can also be generated using the `genscript.exe` utility from the command line interface.

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

```
siebsrv_root/bin or client_root/bin
```

The syntax for running `genscript` is:

```
genscript config_file dest_dir [language]
```

For example:

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

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

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

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

```
siebsrvr_root\webmaster
```

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

```
client_root\public\language_code
```

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

```
siebsrvr_root\webmaster\srfTimestamp\bscripts\all\
```

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

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

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

- Stop and restart the Web server.
- Run the SWE command:

```
<\\host\callcenter\start.swe?SWECmd=UpdateWebImages&SWEPassword=passwd>
```

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

Localization

All the language-specific attributes of Siebel Objects for different languages are maintained in the same repository. Language-specific attributes include translatable strings and locale specific layout information. The Editors in Siebel Tools allow you to edit the locale-specific attributes of objects, such as Applets, Views, and Controls in multiple languages.

Locale Object Types

For all Siebel object types that contain localizable data, such as the Title property of an Applet, there are child locale objects used to define the locale-specific data for the parent object. For example, the Applet object type includes a child object type called Applet Locale.

Locale object types store their data in a set of repository tables used specifically for storing locale-specific data. These tables follow a naming convention that includes the name of the base table followed by the suffix `_INTL`. For example, the Applet object type stores its data in the `S_APPLET` table; the Applet Locale object type stores its data in the `S_APPLET_INTL` table.

Siebel Tools Language Mode

To determine what localizable data to use with translatable attributes, Siebel Tools runs in a particular language mode. Siebel Tools itself runs in an English-American user interface, but you can edit all localizable data in the language that you wish. English-American is the default edit language. Siebel Tools has a language mode that can be set from the Development Tools Options dialog box. See [Figure 41](#).

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

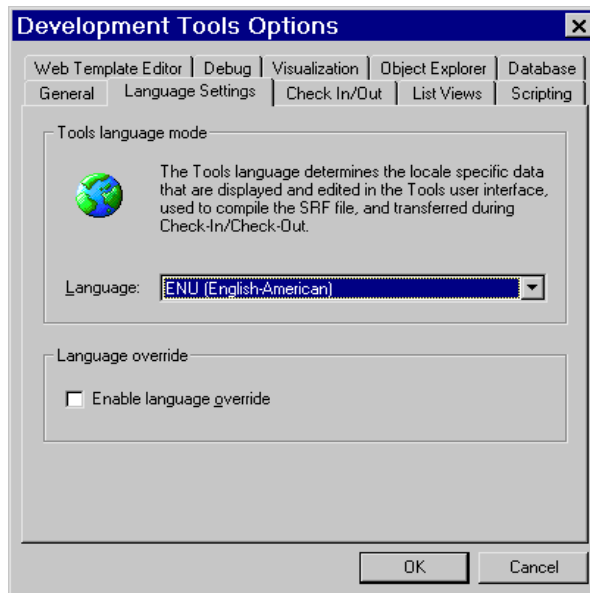


Figure 41. Development Tools Options

Check In/Out

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

Locale Management Utility

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

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

See [Chapter 10, “Logical User Interface Objects Configuration”](#) for more information about using the LMU.

Controlling Visibility Using Siebel Tools

Siebel Tools can be used to control user access to views and data in Siebel applications.

- [“Visibility Establishment Process”](#)
- [“Visibility Property Settings in Siebel Tools” on page 182](#)
- [“Security Considerations” on page 183](#)

Visibility Establishment Process

This section briefly discusses visibility establishment. *Visibility* refers to the level of access users have to the content of the application.

After a successful logon to the DBMS, the Siebel application engine locates the user's database logon ID in the Employee business component in the Siebel applications. The employee records may be viewed and maintained in Siebel applications by anyone with administration access rights in the Application Administration screen.

Each employee record has a Login Name value assigned, which is the same as that person's database username. This value makes it possible for the system to access the user's Employee record following database logon. Each employee record also has a Position field and a Responsibility field. These two fields (each of which can hold multiple values), in addition to the user logon, establish visibility for that user. Visibility refers to the set of access rights that identify the portions of the application and data that are visible to specific employees (users). The roles of Responsibility, Position, and Login Name are explained briefly as follows:

- **Responsibility.** A user can have one or more responsibilities assigned. A responsibility is a class of multiple users who require access to the same set of application features. A user's responsibility (or set of responsibilities, if more than one) identifies the views to which the user can navigate in Siebel applications. When a user's responsibility does not include a specific view, the ability to navigate to that view (using menu options, screen tabs or drilldown features in other views) is disabled.
- **Login Name.** The user's login name is registered in records that the user creates, thereby providing Personal visibility to these records. In some business components, such as Contact, it is also possible for an authorized user to assign Personal visibility to a particular user.
- **Position.** A user can have one or more positions assigned. The position describes the person's job title in the organization, such as Marketing Assistant, Lead Engineer, or Call Center Agent. The user's position may appear in the Sales Team for particular records, which provides Sales Team visibility to those records. Sales Teams are updated through the territory assignment process in Siebel Assignment Manager. The position is also used in determining if the user supervises persons who have visibility to particular records. In this case, the user has Manager visibility to those records.

In addition to visibility rules, which establish the user's access rights to records through a network to which the user is directly connected, there are routing rules. The routing rules specify which records are to be propagated to mobile users. For example, routing rules may limit a particular user to receiving only certain accounts, thereby eliminating the unnecessary transmission of records for which the user has no need.

For further information about authentication and routing rules, see *Security Guide for Siebel eBusiness Applications* and *Siebel Remote and Replication Manager Administration Guide*.

Visibility Property Settings in Siebel Tools

Access control in Siebel applications is implemented through three mechanisms: responsibilities, visibility, and routing.

Record access visibility behavior is controlled by properties in object definitions that you can modify. The object types that have visibility-related properties are briefly described in [Chapter 7, “Business Objects Layer.”](#) Routing rules are briefly discussed in the section on Dock Objects Wizard in [Chapter 6, “Adding Custom Extensions to the Data Model.”](#) For further information about authentication and routing rules, see *Security Guide for Siebel eBusiness Applications* and *Siebel Remote and Replication Manager Administration Guide*.

Security Considerations

The following section briefly discusses security issues related to your Web application. For further details on secure views and login, see *Security Guide for Siebel eBusiness Applications*.

Secure Views

You can create secure views for your Web application, using the HTTPS protocol. If a view is marked as secure, the Siebel Web Engine will verify that the current request used the HTTPS protocol, thereby preventing a client from obtaining access to a secure view by typing HTTP instead of HTTPS into their browser.

To specify that a view is secure

- 1 Edit the Secure attribute of the View object in Siebel Tools.

By default, this attribute is FALSE.

- 2 To make the view secure, set this attribute to TRUE.

If a view is secure, all URLs to the view generated by the Siebel Web Engine will specify the HTTPS protocol.

NOTE: The implementation of HTTPS is external to Siebel Web Engine. HTTPS is negotiated by the browser and the Web Server. Siebel Web Engine only specifies that HTTPS should be used for a particular view. Therefore, any server that is expected to provide secure views must have HTTPS enabled.

Explicit Login

You can specify that users must type in their password and username to access a view, if they have not already done so.

Users can log in to Siebel Web Engine applications using a cookie (after having selected Save My Username and Password), or by explicitly typing their username and password at the login page. If they have logged in using a cookie, you may still want them to supply their username and password to access a sensitive part of the Web site.

To specify that you want a view to require a login

- 1 Edit the Explicit Login attribute of the View object in Siebel Tools.

By default, this attribute is FALSE.

- 2 To require login for a view, set this attribute to TRUE.

In the case that the user logs in using a cookie, and they attempt to access an explicit login view, they will be required to explicitly type their username and password at the login screen before gaining access to the view. Users will only be required to do this once per session. After supplying their username and password, all subsequent visits to the explicit login view will not require login.

User Authentication

Authentication is the process of verifying the identity of a user before allowing the user to access an application. Siebel applications support three approaches for authenticating users: database authentication, security adapter authentication, and Web single sign on.

Security adapter authentication and Web single sign on are external authentication strategies. You can implement either strategy with a Siebel-provided security adapter or Siebel-compliant third party security adapter and other third party software.

Security Guide for Siebel eBusiness Applications is the principal resource on the *Siebel Bookshelf* for detailed information about implementing user authentication strategies, registering and administering users, and controlling user access to data for Siebel employee, partner, and customer applications.

Other Ways to Customize Application Behavior

Although the purpose of this reference guide is to permit extensive application configuration, it is sometimes not necessary to use Siebel Tools to achieve the desired results.

Siebel applications make use of several mechanisms that support business logic, such as Siebel Personalization, Siebel SmartScript, Siebel Business Process Designer, Siebel Assignment Manager, State Model, and Siebel ePricer. These areas of functionality are controlled through client administration views rather than Siebel Tools, but are generally important for developers as well as administrators.

Personalizing Your Web Application

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

Some key points about personalization:

- Personalization is available on any applet in your Web application.
- Personalization uses rules and rule sets to specify which records a user should see in a given applet, based on the user's profile. Rules evaluate profile attributes to determine which records and applications to display. A rule set is a group of rules. If desired, you can create multiple rule sets such that if none of the criteria in one set of rules is met, the next rule set is evaluated.
- The user profile is based on any attribute that belongs to a Contact and the contact's account (if the user is a contact) or any attribute that belongs to an Employee and the employee's division (if the user is an employee).
- Personalization uses a new object called the User Profile Attributes to hold and retrieve elements of a user profile. These attributes can be used for display in the user interface of the Web application, and in Rules that determine the content users see.
- You can track events that occur in the context of the Web application. Specifically Siebel Personalization can track application, business component and applet events. When an event occurs, it triggers a Personalization Action which modifies a user's profile attributes.

- Actions can be called by a rule or on an event. An action is used to set either a predefined profile attribute or a profile attribute created dynamically during run time. Profile attributes created dynamically during run time only exist for the duration of the user session. Profile attributes that are configured in Siebel Tools and those that are created during run time can be used to store state information in much the same way that variables stored in cookies or persistent frames might be. Wherever possible, profile attributes should be used in placed of cookies.
- Rules or actions can invoke business component methods or business services methods. Typically, these methods are used to return values that can be used either as criteria for a rule, or for setting a profile attribute.

For more information about personalization, refer to *Personalization Administration Guide*.

Managing Web Content with Siebel eBriefings

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

For more information about eBriefings, refer to *Siebel eBriefings Guide*.

Dynamic Data Capture with Siebel eSmartScript

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

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

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

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

For more information about Siebel eSmartScript, refer to *Siebel SmartScript Guide*.

Siebel Assignment Manager

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

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

Siebel Business Process Designer

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

State Model

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

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

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

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

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

Siebel ePricer

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

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

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

Siebel ePricer is composed of the following components:

- Price lists. Price lists contain base prices.
- Pricing models. Pricing models are a management tool to control a set of related pricing factors.
- Pricing factors. Pricing factors are statements that define conditions and pricing adjustments.
- Scripting. The scripting capability lets you use business services with a pricing factor to extend the pricing calculation and to access external data.
- Pricing validation. The validation facility lets you test pricing factors and the pricing model before releasing for use by end users.
- Reports. The reporting facility lets you print reports of pricing factors.
- Pricer Engine. The Pricer engine evaluates conditional statements and applies pricing adjustments.

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

Application Configuration (Basic Concepts)

Other Ways to Customize Application Behavior

This chapter describes the objects in the data layer of the Siebel application architecture.

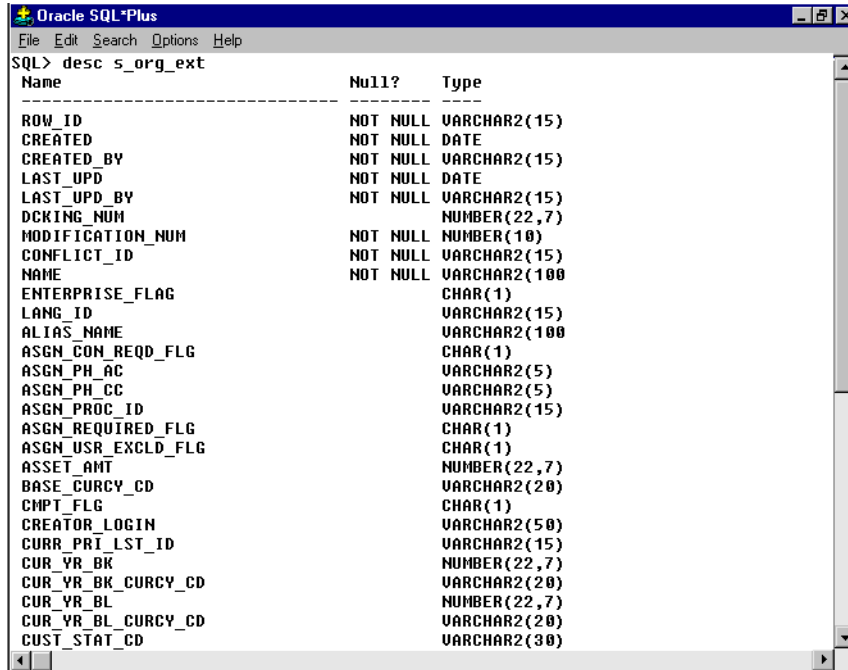
Data Object Types

Data objects fall under two data object types:

- Table, and child object types:
 - Column
 - Index
 - User Key
- EIM (Enterprise Integration Manager) objects

Interface Table data objects map to underlying physical data objects that are stored in a relational DBMS.

Siebel applications store and retrieve most of the data from a relational DBMS. [Figure 42](#) shows the database columns in a Siebel application database table, S_ORG_EXT, as displayed in the SQL*Plus viewer in Oracle.



The screenshot shows the Oracle SQL*Plus interface with the command 'SQL> desc s_org_ext' entered. The output displays the table's structure with columns, nullability, and data types.

Name	Null?	Type
ROW_ID	NOT NULL	VARCHAR2(15)
CREATED	NOT NULL	DATE
CREATED_BY	NOT NULL	VARCHAR2(15)
LAST_UPD	NOT NULL	DATE
LAST_UPD_BY	NOT NULL	VARCHAR2(15)
DCKING_NUM		NUMBER(22,7)
MODIFICATION_NUM	NOT NULL	NUMBER(10)
CONFLICT_ID	NOT NULL	VARCHAR2(15)
NAME	NOT NULL	VARCHAR2(100)
ENTERPRISE_FLAG		CHAR(1)
LANG_ID		VARCHAR2(15)
ALIAS_NAME		VARCHAR2(100)
ASGN_CON_REQD_FLG		CHAR(1)
ASGN_PH_AC		VARCHAR2(5)
ASGN_PH_CC		VARCHAR2(5)
ASGN_PROC_ID		VARCHAR2(15)
ASGN_REQUIRED_FLG		CHAR(1)
ASGN_USR_EXCLD_FLG		CHAR(1)
ASSET_AMT		NUMBER(22,7)
BASE_CURCY_CD		VARCHAR2(20)
CMPT_FLG		CHAR(1)
CREATOR_LOGIN		VARCHAR2(50)
CURR_PRI_LST_ID		VARCHAR2(15)
CUR_YR_BK		NUMBER(22,7)
CUR_YR_BK_CURCY_CD		VARCHAR2(20)
CUR_YR_BL		NUMBER(22,7)
CUR_YR_BL_CURCY_CD		VARCHAR2(20)
CUST_STAT_CD		VARCHAR2(30)

Figure 42. S_ORG_EXT Table Displayed in Oracle's SQL*Plus

Figure 43 shows the column object definitions in the S_ORG_EXT table object definition in the Object List Editor of Siebel Tools.

Name	Type	Changed	Inactive	User Name
ALIAS_NAME	Data (Public)			Alias Name
ASGN_CON_REQD_FLG	Data (Public)			Assignment Contact Required Flag
ASGN_PH_AC	Data (Public)			Assignment Phone Ac
ASGN_PH_CC	Data (Public)			Assignment Phone Cc
ASGN_PROC_ID	Data (Public)			Assignment Process
ASGN_REQUIRED_FLG	Data (Public)			Assignment Required Flag
ASGN_USR_EXCLD_FLG	Data (Public)			Assignment User Excluded Flag
ASSET_AMT	Data (Public)			Asset Amount
BASE_CURCY_CD	Data (Public)			Base Currency
CMPMT_FLG	Data (Public)			Competitor Flag
CONFLICT_ID	System			Conflict Id
CREATED	System			Created
CREATED_BY	System			Created By
CREATOR_LOGIN	Data (Public)			Creator Login Name
CURR_PRI_LIST_ID	Data (Public)			Curr Pri List Id
CUR_YR_BK	Data (Public)			Current Year Booked Revenue
CUR_YR_BK_CURCY_CD	Data (Public)			Current Year Book Currency Code
CUR_YR_BL	Data (Public)			Current Year Billed Revenue
CUR_YR_BL_CURCY_CD	Data (Public)			Current Year Book Currency Code
CUST_STAT_CD	Data (Public)			Customer Status Code
DCKING_NUM	System			Docking Number
DESC_TEXT	Data (Public)			Description

Figure 43. S_ORG_EXT Columns in the Object List Editor of Siebel Tools

One Table object definition exists in Siebel Tools for each database table, and there are similar correspondences between Column and Index object definitions in Siebel Tools with physical columns and indexes.

Tables

A table object definition is the direct representation of a physical database table in a DBMS.

Siebel applications provide a set of standard tables that are included in standard Siebel applications. These tables have predefined names and structures, and typically begin with one of the following prefixes:

- **S_.** Table names starting in S_ are standard tables for supplied with Siebel Sales and Siebel Service. For example, the S_CONTACT table stores contact information, and S_OPTY stores opportunity information. Nearly all standard tables are of this type.
- **W_.** Table names starting in W_ are Data Warehouse tables used in Siebel Analytics to denormalize data used in the S_ tables.

Base Tables

The term *base table* is used in two different contexts:

- The base table for an extension table is the table it extends. This is specified in the Base Table property of the extension table's object definition. Extension tables are discussed in [“Extension Tables” on page 196](#).
- The base table for a business component is the table that provides most of its essential fields. This is specified in the Table property of the Business Component object definition. The set of fields supplied by the base table is supplemented by nonupdateable fields that are obtained from joins. Business components are discussed in [“Business Components” on page 342](#).

Tables have various styles based on the value in the Type property. They include, among others, Data, Extension, Intersection, and Interface. Table styles are summarized under the Type property heading in [“Properties of the Table Object Type” on page 195](#), and each is discussed in detail in a subsequent section.

Properties of the Table Object Type

The following are the key properties in a table object definition:

- **Name.** Provides the name of the table in the DBMS.
- **Type.** Indicates which of the following styles describes the table.
 - **Data (Public).** Public data tables are among the original set of tables implemented in Siebel applications. They hold data that is made available through business components to developers and users. Public data tables can be extended using extension tables and extension columns. These extension are subject to database restrictions.

Data tables are discussed in [“Data Tables” on page 196](#).

- **Data (Private).** Private data tables are similar to public data tables, but cannot have extension columns.
- **Data (Intersection).** Identifies an intersection table. An intersection table implements a many-to-many relationship between two data tables.

Intersection tables are discussed in [“Intersection Tables” on page 204](#).

- **Extension.** An extension table adds additional columns to a data table that the original data table is unable to hold due to DBMS platform or Siebel application design restrictions. Extension table names have an `_X` suffix, or `_XM` or one-to-many, or `_T` for TAS extension tables.

NOTE: Extension tables that have `_XM` suffix have table type of “Data (Public).”

Extension tables are discussed in [“Extension Tables” on page 196](#).

- **Interface.** Interface tables are used by Siebel Enterprise Integration Manager (EIM) to import initial data for populating one or more base tables and subsequently to perform periodic batch updates between Siebel applications and other enterprise applications. Interface table names end in `_IF` or `_XMIF`.

Interface tables are discussed in [“Column Objects” on page 215](#).

- **Database View, Dictionary, Journal, Log, Repository, Virtual Table, and Warehouse styles.** These are all table types that are reserved for Siebel internal use.

- **Extension (Siebel).** These tables are reserved for Siebel use only. They are usually extensions from S_PARTY. If customers want to extend person- and organization-related tables they need to extend from S_PARTY.

For example, S_CONTACT is an extension table of S_PARTY. Because S_CONTACT is of type Extension (Siebel), you cannot use it as a parent table for an extension table. You must use S_PARTY.

For a business component based on your new table to show data from S_CONTACT, you must create a Join object that references S_CONTACT and has a Join Specification child object with a Source Field property set to Parent Id and Destination Column property set to ROW_ID. The row ID of an S_CONTACT record will be the same as the row ID of the corresponding S_PARTY record.

- **Base Table.** Identifies the base table if the table in the object definition is an extension table. If the table in the object definition is a base table, this property is blank. An extension table always identifies a base table.
- **User Name.** A longer, descriptive name that aids in identifying the table when used in configuration.
- **Alias.** A name that can be used as a synonym for the table name to make the name more understandable. For example, an alias such as S_Organization_External could be specified for the S_ORG_EXT table.

Data Tables

Data tables comprise most of the tables in Siebel applications. They serve as base tables for business components, and their columns provide the data for fields. Data tables can be public or private.

Extension Tables

An *extension table* provides additional columns to a data table that cannot be directly added to the original table because the underlying DBMS may support only a limited number of columns, or will not allow adding a column to a table once it is populated with data. An extension table allows you to provide additional columns for use as fields in a business component without violating DBMS or Siebel application restrictions.

An extension table is a logical augmentation of an existing table. Its columns are provided mostly for developers, and are generally not used by standard Siebel applications. An extension table extends a base table in the sense that it effectively adds additional columns. These columns are not physically part of the base table, but are available for use in a business component alongside the base table columns as if they were.

When columns in a base table are updated, the time stamps of its extension tables are not updated unless columns in those extension tables are also updated.

When records in an extension table are changed, system columns in a parent table are updated. This is done because the associated record in an extension table is considered by the object manager to be logically a part of its parent record.

The relationships between a base table, an extension table, and the business component that uses them are illustrated in [Figure 44](#).

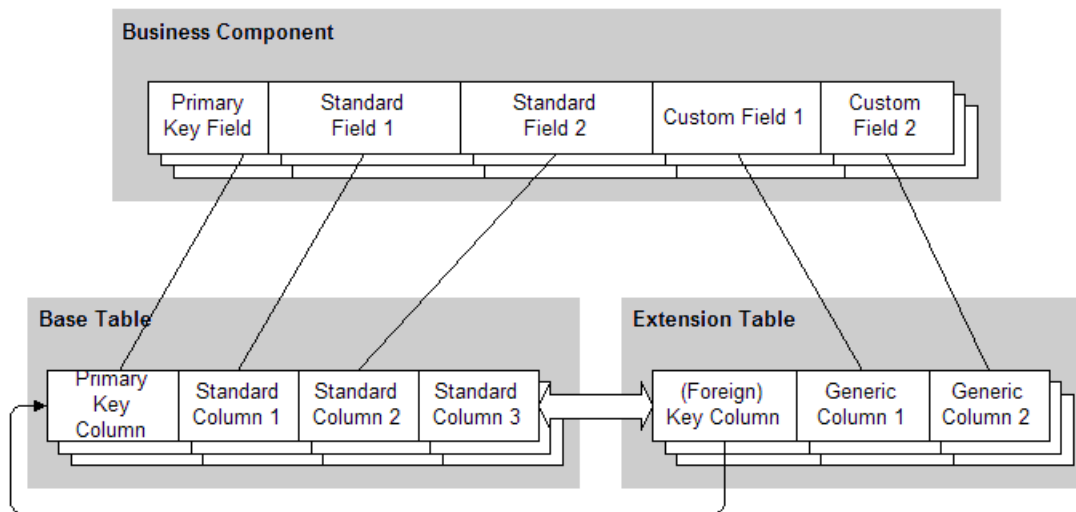


Figure 44. Business Component, Base Table and Extension Table

Note the following distinctions between standard and custom extension tables:

- Siebel applications provide *standard extension tables* for several of the standard data tables. A standard extension table has a predefined relationship with a standard data table. This relationship allows you to add columns for new functionality without making alterations to the base table. You cannot create or delete standard extension tables.
- You can use the Table Wizard to create *custom extension tables* to extend data tables, provided the data tables are of type Data (Public).

An extension table, whether standard or custom, provides a set of generic columns of various data types and lengths for your use. These may eliminate the need to add a custom column to the extension table. Generic columns in an extension table have names of the form `ATTRIB_xx`, where `xx` stands for a two-digit number. For example, there are generic columns named `ATTRIB_04` and `ATTRIB_12`.

The standard Siebel applications use certain columns in extension tables. The following columns in these tables are used:

- **S_CONTACT_X.** `ATTRIB_03`, `04`, `05`, `06`, `07`, `08`, `14`, `15`, `26`, `48`, and `MODIFICATION_NUM`.
- **S_EMPLOYEE_X.** `ATTRIB_48`.
- **S_OPTY_X.** `ATTRIB_04`, `05`, `08`, `09`, `10`, `11`, `15`, `16`, `17`, `18`, `34`, `35`, `36`, `37`, `38`, `39`, `41`, `42`, `43`, `44`, `45`.
- **S_ORG_EXT_X.** `ATTRIB_01`, `02`, `03`, `08`, `14`, `15`, `16`, `27`, `48`, `49`, `50`, `51`, `52`, `53`.

NOTE: Extension columns used by standard Siebel applications should be treated as data columns in base tables—that is, they should not be modified or deleted.

Figure 45 illustrates how columns from a standard one-to-one extension table are used in the Contact business component in Siebel applications.

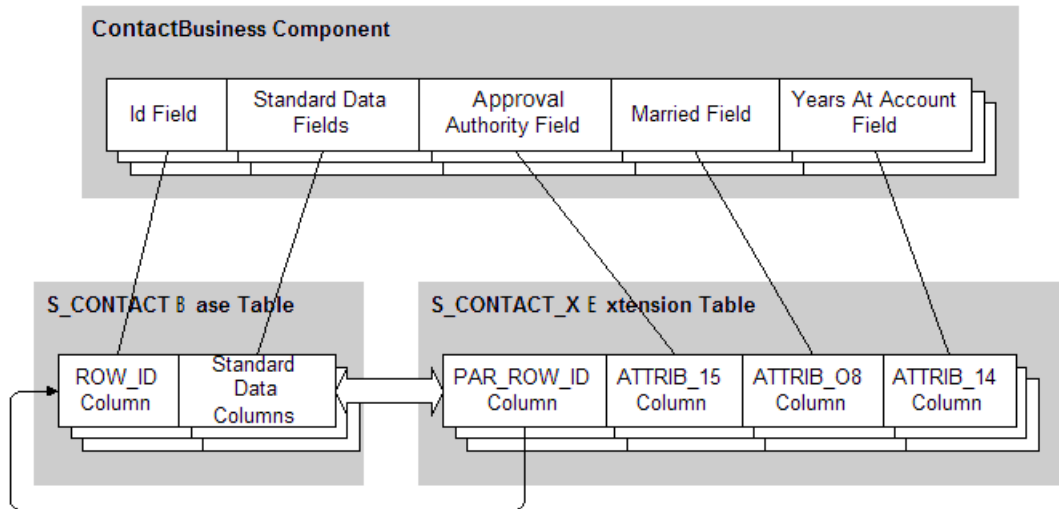


Figure 45. Extension Table Example

There are eight fields in the Contact business component displaying data from generic columns in S_CONTACT_X; only three are shown here. Extension tables themselves are “sparse”—extension table rows exist only for those base table rows that have extension data to store.

Extension tables can be of the one-to-one or one-to-many style:

- Rows in *one-to-one* extension table have a one-to-one relationship with corresponding rows in the base table. A one-to-one extension table extends the base table horizontally, as shown in [Figure 45 on page 199](#). One-to-one extension tables are described in greater detail in the following paragraph.
- In a *one-to-many* extension table, there are multiple extension table rows for each base table row. There are standard one-to-many extension tables for certain of the major business components, including Opportunity, Contact and Account. These are used primarily to create multi-value groups based on user-created business components. One-to-many extension tables are described in [“One-to-Many Extension Tables” on page 203](#).

One-to-One Extension Tables

One-to-one extension tables have the `_X` suffix on their names (with the exception of TAS tables, which have the suffix `_T`). The details of the object definition relationships (excluding the implied join) are illustrated in [Figure 46](#).

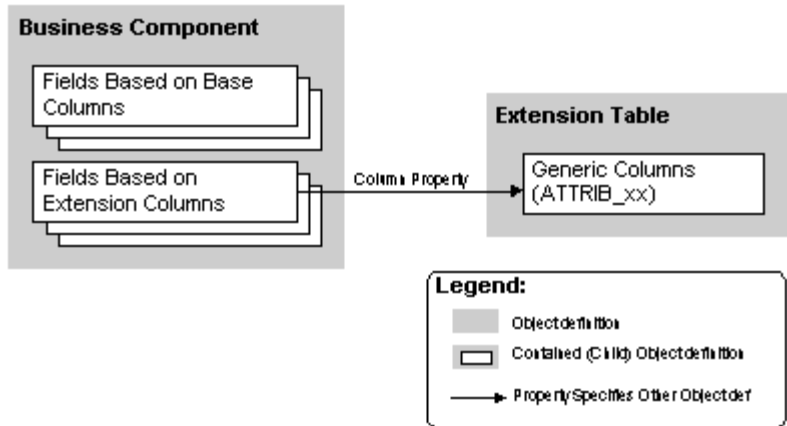


Figure 46. Extension Table Details (Excluding Implied Join)

The object definitions in [Figure 46](#) are:

- **Business component.** Business component being extended.
- **Fields based on base columns.** Fields that represent data from columns in the business component's base table. They are unaffected by the extension table.
- **Fields based on extension columns.** Represent data from columns in the extension table.
- **Extension table.** Provides columns that may be used to add developer-defined fields to the business component.

When writing data to a one-to-one extension table, at least one column of the extension table must be updated for a record to be written to the extension table.

For example, you might want to create a workflow policy based on a column in a 1:1 extension table. If there is no data to be written, the record will not be updated. Therefore, the workflow policy will not be triggered.

Implied Joins

Underlying the one-to-one extension table's relationships with the base table and business component is a set of hidden relationships called an *implied* or *implicit join*. The implied join makes the extension table rows available on a one-to-one basis to the business component that uses the extension table. Every extension table has an implied join with the business component it extends. This join always has the name of the extension table.

An implied join is different from joins defined as object definitions. Data can be updated through an implied join. Data can be displayed only through other joins. This update capability is important for extension table functionality.

When a field in the business component is based on a column in the extension table, the Column property of the Field object is set to the name of the column, and the Join property is set to the name of the extension table. For example, the Birthday field in the Contact business component has a Column property value of ATTRIB_26 and a Join property value of S_CONTACT_X.

The details of the object definition relationships in an implied join are illustrated in Figure 47.

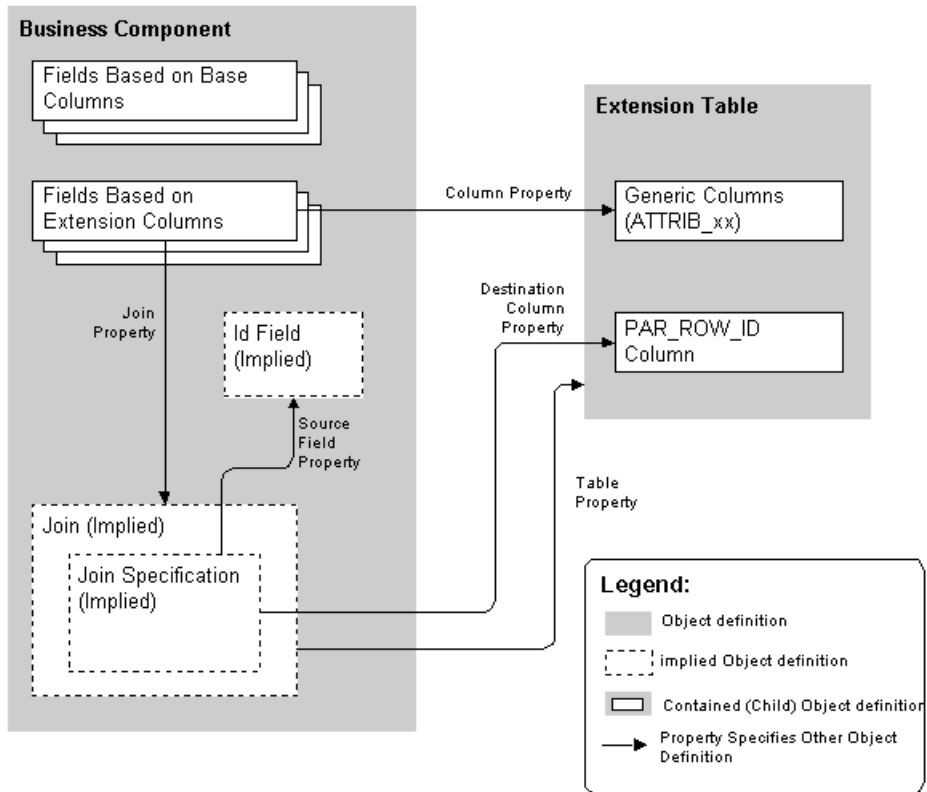


Figure 47. Extension Table Details with Implied Join

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

- **Id field.** The Id field is a system field in the business component. It represents the ROW_ID column in the base table, and it can be used in joins involving extension tables and other joined tables.

- **PAR_ROW_ID column.** PAR_ROW_ID stands for parent row ID. Every extension table has this column, and every extension table row has a value there. It is used as a foreign key to the base table that is extended by the extension table.

For more information, see [“Joins” on page 378](#).

One-to-Many Extension Tables

One-to-many extension tables have a Type property value of Data (Public) rather than Extension. However, from a functional standpoint, one-to-many extension tables are considered extension tables, and they have the same set of generic and system columns. The names of one-to-many extension tables have the suffix `_XM`.

You can extend existing one-to-many extension tables. You can also add new one-to-many extension tables using Advanced Database Extensibility. For information, see [“Advanced Database Extensibility” on page 292](#).

You can use one-to-many extension tables to create multi-value groups and master-detail views that are based on custom business components—that is, business components not present in standard Siebel applications.

For example, you may have a need for three new multi-value fields in the Contact business component to store information on hobbies, prior companies, and areas of expertise for each contact. No business components exist for these entities. However, you can implement the same functionality using `S_CONTACT_XM`, the one-to-many extension table that extends the Contact business component. A one-to-many, rather than one-to-one, extension table is required because there can be many hobbies, prior companies, or areas of expertise for one contact. Since the relationship is one-to-many rather than one-to-one, a link is required rather than an implicit join. See [Chapter 7, “Business Objects Layer”](#) for a discussion of master-detail business components.

Intersection Tables

An *intersection table* implements a many-to-many relationship between two business components.

NOTE: You might find it helpful to read the section titled “Links” on page 393 before reading this section.

A *many-to-many relationship* is one in which there is a one-to-many relationship from either direction. For example, there is a many-to-many relationship between Opportunities and Contacts. One Opportunity can be associated with many Contact people, and one Contact person can be associated with many Opportunities. Two different views can appear (in different business objects) which associate the two business components in opposite ways, as illustrated in Figure 48 and Figure 49 on page 205.

Figure 48 shows the Account Detail - Contacts View, in which one account is displayed with multiple detail contacts.

New	Last Name	First Name	Mr Ms	Work Phone #	Job Title	Email
	Ahl	Avram	Mr.	(773) 555-9870	Vice President	a.ahl@3Com.com
	Genever	Alan	Mr.	(773) 555-6500	Manager, Account F	alan.genever@3Com.com
	Helena	Kristy	Ms.	(773) 555-2223	Supervisor, Logistic	kristy.helena@3Com.com
✓	Ishii	Jill	Ms.	(773) 555-5312	Account Manager	jill.ishii@3Com.com
	Kreisberg	Bob	Mr.	(650) 766-8767	Programmer	bob.kreisberg@3Com.com
	Mann	Elisa	Ms.	(773) 355-8675	Service Manager	elisa.mann@3Com.com
	Navarro	Alan	Mr.	(773) 555-6541	Programmer	alan.navarro@3Com.com

Figure 48. Account Detail - Contacts View

Figure 49 shows the Contact Detail - Accounts View, in which one master contact is displayed with multiple detail accounts.

The screenshot displays the Siebel CRM interface for the 'Contact Detail - Accounts View'. The top section is a form for the 'Contact master record' for Gina Aamot. Below the form is a navigation bar with tabs for 'More Info', 'Accounts', 'Activities', 'Activity Plans', 'Addresses', 'Agreements', 'Assets', 'Attachments', 'Calendar', and 'Campaigns'. The 'Accounts' tab is active, showing a table of 'Accounts detail records'.

Primary	New	Name	Site	Main Phone #	Territories	Industries	Status	UF
Y		AEP Communication	Columbus, OH			electric services		www.
N	*	Murphy Brewery Irel	Cork/Ireland	+35321503371			Current Customer	

Figure 49. Contact Detail - Accounts View

To implement a many-to-many relationship, two links and a table designated as an *intersection table* are required. The table is designated as an intersection table in its Type property by means of a value of Data (Intersection). The intersection table represents the many-to-many relationship as two one-to-many relationships, which the underlying DBMS is designed to handle. There is no database construct that implements many-to-many relationships directly. This representation design is illustrated in [Figure 50](#).

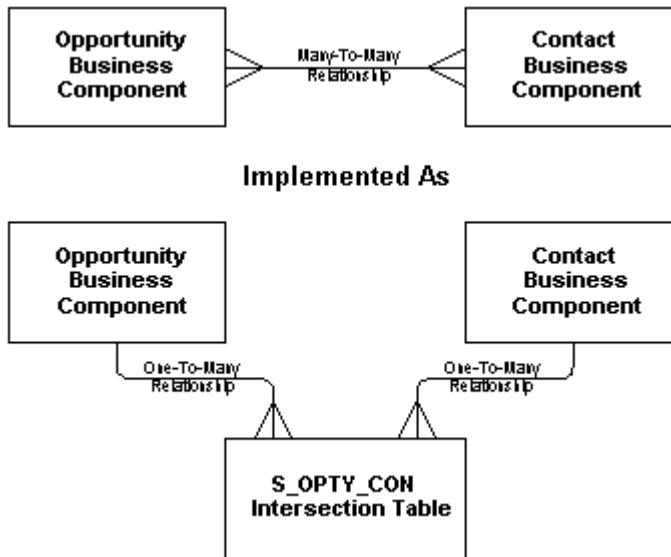


Figure 50. Many-to-Many Relationship as Two One-to-Many Relationships

You can configure custom intersection tables using Advanced Database Extensibility. For information, see [“Advanced Database Extensibility” on page 292](#). However, if your organization needs this functionality, contact Siebel Expert Services for assistance.

How Intersection Tables Are Configured

Figure 51 displays the object types used in the implementation and use of an intersection table.

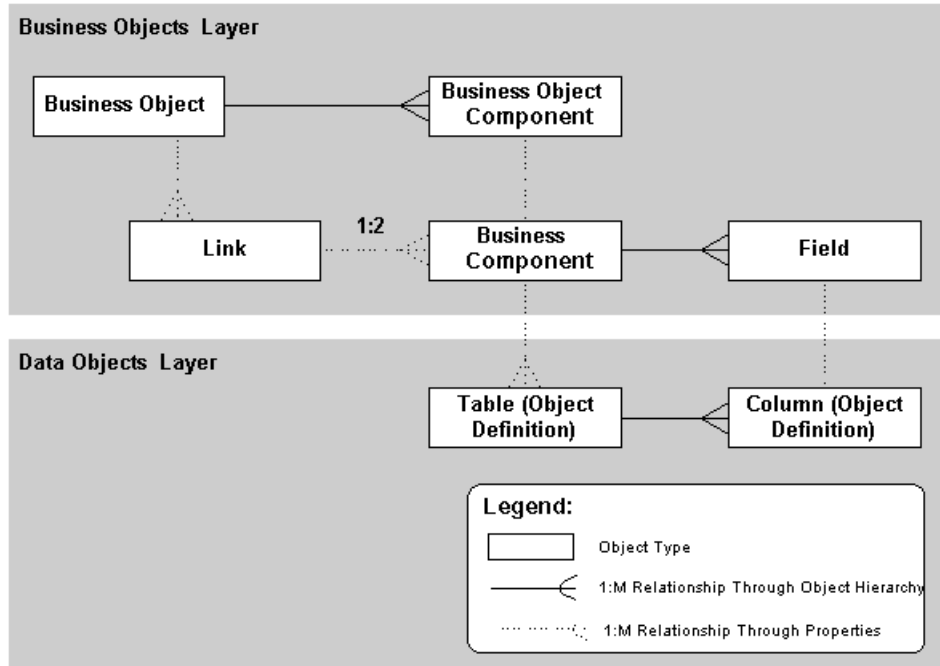


Figure 51. Intersection Table Architecture

The intersection table contains one row for each association between a row in one business component's base table and a row in the other business component's base table, regardless of which one-to-many relationship the association pertains to. The association row in the intersection table stores the ROW_ID values of the row in each business component base table. The details of intersection table relationships are illustrated in Figure 52.

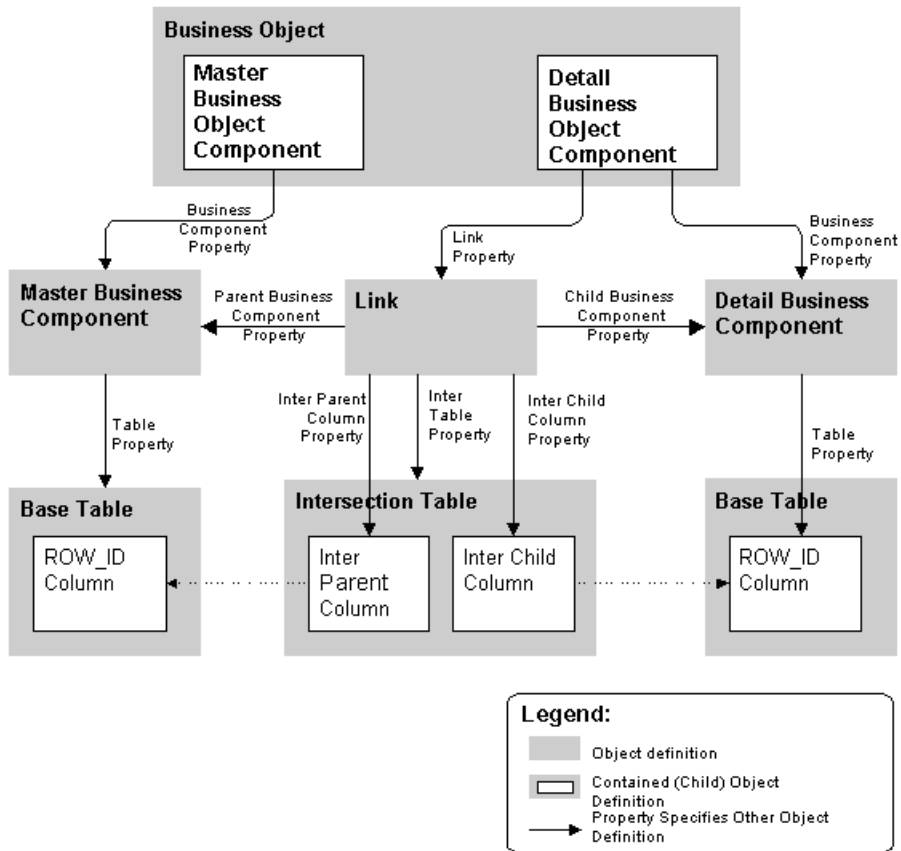


Figure 52. Intersection Table Details

[Figure 52](#) shows the object definition relationships in greater detail.

Notice how the associations stored in one intersection table serve both the Opportunity/Contact and Contact/Opportunity links, and their corresponding views. An association is simply a pair of ROW_ID values pointing to rows in their respective business component base tables. One association may appear in both views, for example, the association between Cynthia Smith and Smith Dry Goods in [Figure 52 on page 208](#).

The set of object definitions and relationships in [Figure 52 on page 208](#) pertains to one of the two links. The other link uses the same set of object types, but slightly different relationships.

The following are descriptions of the object definitions in [Figure 52 on page 208](#):

- **Business object.** The business object references the link (indirectly through the business object's child business object component) that uses the intersection table. It also contains the two business components included in the link.
- **Business object components.** Business Object Component object definitions are used to include business components in the business object. Business Object Component is a child object type of Business Object. The detail business object component references both the detail business component, by means of the Business Component property, and the link, by means of the Link property. The master business object component only references its corresponding business component.
- **Link.** The link object definition establishes a one-to-many relationship between the two business components in a particular direction. That is, the property settings in the link specify that one business component is the master and the other is the detail in the master-detail relationship.
- **Master and detail business components.** The two business components are specified in the link. They provide data to the user interface object definitions that display the master-detail relationship. The base table of each business component contains the ROW_ID column referenced by the Inter Child Column (detail) and Inter Parent Column (master) properties of the Link object type.

- **Intersection table.** The intersection table holds the associations between rows in the base tables of the master and detail business components. Each row in the intersection table represents one association between the two business components. Two columns in the intersection table serve as foreign keys to the base tables of the two business components. These columns are identified in the Inter Parent Column and Inter Child Column properties of the link.
- **Inter Parent column.** This column in the intersection table holds the pointer to the associated row in the master business component's base table. It is identified in the Inter Parent Column property of the Link object.
- **Inter Child column.** This column in the intersection table holds the pointer to the associated row in the detail business component's base table. It is identified in the Inter Child Column property of the Link object.
- **ROW_ID columns.** The base table of each business component has a unique identifier column for the rows in that table. This is the ROW_ID column.

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

Figure 53 illustrates the property settings in the two links used to implement a many-to-many relationship—in this case the relationship between Opportunities and Contacts.

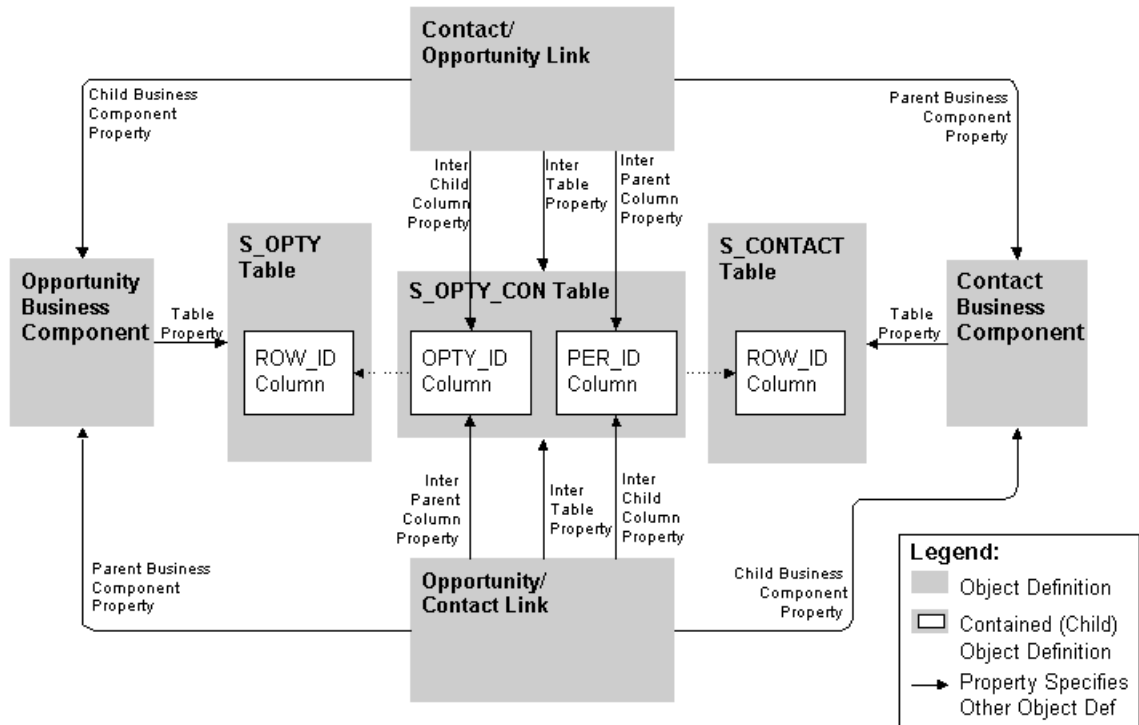


Figure 53. Two-Link Intersection Table Example

Notice how the inter child column of one link is the inter parent column of the other, and the other way around. Also notice how the parent business component in one link is the child business component in the other, and the other way around. The two links are mirror images of each other.

Intersection Data in the Intersection Table

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

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

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

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

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

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

Joins are not the only way to expose intersection data. An alternative is to use the intersection table as the base table for an intersection business component. Intersection business components are described in [“Intersection Business Components” on page 356](#).

Updating Fields That Are Based on Columns in Extension Tables of Intersection Tables

It is not possible to update a field that is based on a column in an intersection table's standard extension table, through an implied or explicit join from the parent or child business component (Figure 54).

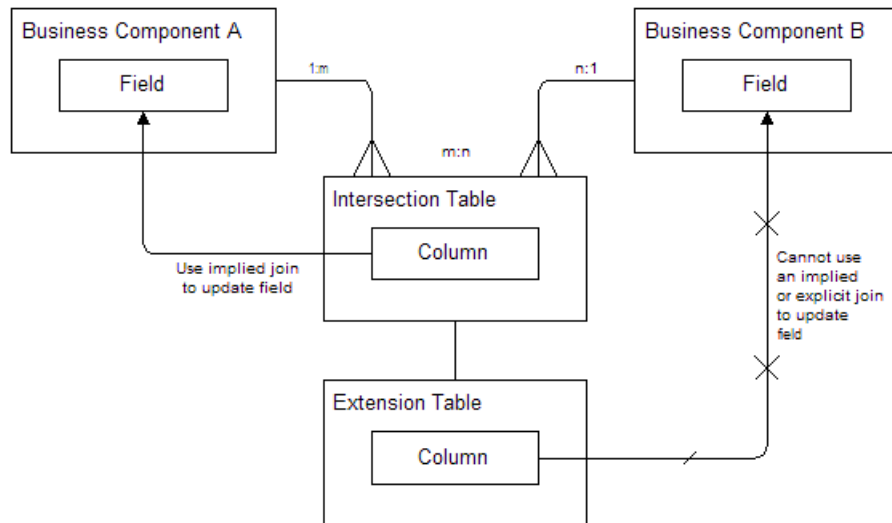


Figure 54. Cannot Use an Implied or Explicit Join to Update the Field

To update such fields, you can create a field in an intersection business component (Figure 55). Expose this field in the parent or child business component (Business Component B in Figure 55) using a multi-value link and multi-value field.

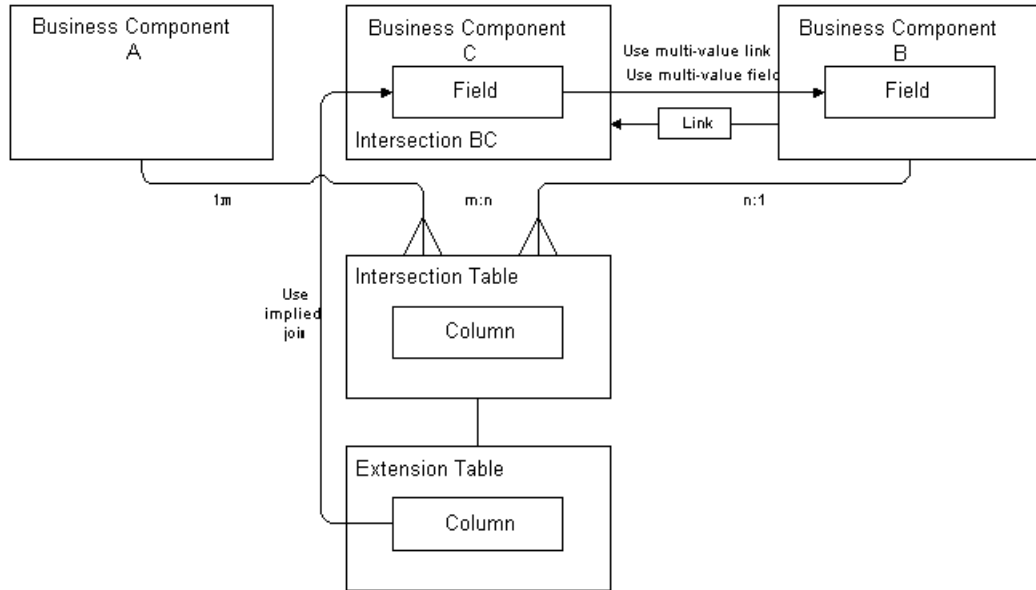


Figure 55. Use a Field in an Intersection Business Component with a Multi-Value Link and a Multi-Value Field

However, an easier solution is to use a custom extension column to the intersection table.

Column Objects

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

A column has one of several styles based on the value in the Type property. These styles include Data, Extension, IFMGR, System, and others.

The Column object type is described in the next section followed by discussion of the various column styles.

Column Object Type

The column object corresponds to one database column in the database table that is represented by the parent table object definition. Each database column in the database table needs to have a corresponding column object definition. The important properties of the Column object type are as follows:

- **Name.** Provides the name of the database column in the database table.
- **Default.** Provides a default value when new rows of this table are added.
- **Physical Type (Physical Type Name).** Identifies the data type of the column in the database. The following data types are supported:
 - **Character.** Used for fixed-length text. Also used for Boolean columns, which are character columns with a length of 1.

NOTE: Defining a Column as a Char when the data being stored in the column is variable in length may cause the data to be padded with blank spaces in order to make up the full size of the length of the column. This may cause problems in Siebel Remote.

- **Long.** Long text. You can store approximately 16K worth of data in long columns. By default, you cannot have char greater than 1. If you want, you need to set the preference under Options/Database.
- **Varchar.** Variable-length text. Used for memo-type fields and to store row-ID and foreign key values.
- **Number.** Any numeric data. Typical numeric columns in Siebel applications are 22,7 for general-purpose numbers, and 10,0 for integers.

Data of this type is limited to 16 digits without a decimal point or 15 digits before a decimal point.

- **UTC Date Time.** Time is saved in Greenwich mean time.
- **Date.** Date values only, without time.
- **Date Time.** Combined date and time values in the same column.
- **Time.** Time values only, without the date.
- **Precision.** Specifies the maximum number of digits in a number column. For noninteger columns, the precision is 22. For integer columns, the precision is 10.
- **Scale.** Specifies the maximum number of digits after the decimal point. For noninteger columns, the scale is 7. For integer columns, the scale is 0.
- **Primary Key.** If TRUE, this column is the primary key for the table. With minor exceptions, the ROW_ID column in a table is the primary key, and has a TRUE value for this property.
- **Type.** Indicates which of the following styles describes the column:
 - **Data (Public).** Public data columns are among the original set of columns implemented in Siebel applications. They hold data that is made available through fields to developers and users.
 - **Data (Private).** Private data columns are reserved for Siebel use only; they apply to tables used to manage the EIM process (for example, interface tables).
 - **Denormalized.** A denormalized column duplicates the data in a column in another (base) table for performance reasons. The table and column names of the duplicated column are specified in the Denormalization Path property of the Column object definition of the denormalized column. The denormalized column should not be in the same table as the column whose data it duplicates.
 - **Extension.** An extension column is generally not used by standard Siebel applications. It is used only in reconfigured applications. There are three kinds of extension columns: standard extension columns, custom extension columns in a base table, and custom extension columns in an extension table.

- **IFMGR: xxx.** These columns have names such as IFMGR: ROW_ID and IFMGR: Status. They are found in interface tables, and are for internal use by the Siebel Enterprise Integration Manager.

NOTE: Interface tables also contain special columns, such as IF_ROW_STAT and IF_ROW_BATCH_NUM. These columns are related to EIM processing, but you can modify the contents of these columns. They have a type of System rather than IFMGR: xxx.

- **System.** System columns appear in all tables in Siebel applications. However, no one set of system columns appears in every table. You can use the data in system columns for various purposes, although most system columns are read-only.

NOTE: When configuring a custom extension column, you should set only the following properties: Comments, Default, Foreign Key Table Name, Inactive, LOV Bounded, LOV Type, Name, Unable, Physical Type, Precision, Scale, Text Length, and the Translation Table Name (this property should be set to S_LST_OF_VAL for multilingual list of values).

Data Columns

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

Extension Columns

Extension columns have a value of Extension in their Type property.

NOTE: Always use extension columns. Do not use Siebel-defined columns for other purposes even if they seem to be unused.

An extension column is a column that is not used by standard Siebel applications. There are three kinds of extension columns:

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

Standard Extension Columns

Each extension table provided with Siebel applications includes standard extension columns of various data types. [Table 11](#) lists the different data types found in Siebel extension tables and the number of columns of each data type. These columns are named ATTRIB_ *nn*, where *nn* is a value from 01 to 47.

Table 11. Standard Extension Columns

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

NOTE: Extension columns with a Physical Type of VarChar have a text length limit of 2000.

The benefit of using standard extension columns is that they provide the means to add fields to business components for new functionality with a minimum of effort and database impact. If there is a need for a custom column, you can adapt an existing standard extension column in an existing standard extension table without adding any new columns to the database schema.

However, the standard Siebel applications use certain columns in extension tables. The following columns in these tables are used:

- **S_CONTACT_X.** ATTRIB_03, 04, 05, 06, 07, 08, 14, 15, 26, 48, and MODIFICATION_NUM.
- **S_EMPLOYEE_X.** ATTRIB_48.

- **S_OPTY_X.** ATTRIB_04, 05, 08, 09, 10, 11, 15, 16, 17, 18, 34, 35, 36, 37, 38, 39, 41, 42, 43, 44, 45.
- **S_ORG_EXT_X.** ATTRIB_01, 02, 03, 08, 14, 15, 16, 27, 48, 49, 50, 51, 52, 53.

NOTE: Extension columns used by standard Siebel applications should be treated as data columns in base tables—that is, they should not be modified or deleted.

Extension Columns and Foreign Keys

Use caution when configuring a standard extension column to hold foreign keys; generally you should avoid the practice. Foreign key extension columns can be appropriate when pointing to enterprise-visible business objects, but not when pointing to limited-visibility business objects such as Opportunity, Contact, Account, or Service Request.

Foreign key relationships based on extension columns in the limited-visibility situation can cause some users not to receive the record, causing the loss of the relationship for all users.

For example, a foreign key may be set to 'No Match Row Id' if Siebel is unable to find the parent record that the foreign key is pointing to. Such cases could arise when you have a limited-visibility business object routed to a mobile user based on a set of visibility rules. As not all records are routed to a local database, your client might end up with a situation where there is a record with a Primary Id Field (for example, on a Multi Value Link) pointing to a record not present on their local database. In such cases, Siebel sets this field to 'No Match Row Id'. Subsequently, when the user synchs up with the server, this Primary Id Field will be set to 'No Match Row Id' for all users on the server. Also, business objects can change from enterprise-visible to limited-visibility with a new release of Siebel applications.

You will also need the assistance of Siebel Expert Services to configure EIM to import data into a foreign key column, because the necessary EIM object types are not customer-configurable. For more information on using EIM to populate foreign key columns, see [“EIM Interface Tables” on page 228](#).

If in doubt, avoid configuring extension columns as foreign key columns.

NOTE: Do not define columns with names longer than 18 characters in the DB2 environment.

System Columns

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

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

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

The format of the ROW_ID is one of the following:

CP-NS For records created by the Siebel Sales Enterprise user interface

XX-XX-XXX For records created by Interface Manager

where:

CP = Corporate Prefix, up to 2 alphanumeric characters

NP = Next Prefix, up to 6 alphanumeric characters

NS = Next Suffix, up to 7 alphanumeric characters

Total maximum 15 alphanumeric characters.

NOTE: Do not alter the ROW_ID column. It is unique throughout the database with the exception of when there is an extension table for the base table. In such cases, the ROW_ID column in the extension table is a duplicate of the corresponding ROW_ID column in the base table.

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

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

NOTE: The CREATED, CREATED_BY, LAST_UPD, and LAST_UPD_BY columns all provide date-time and logon stamps for record creation and update on the client, not server, databases.

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

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

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

Updating the extension table does not update the base table automatically. This needs to be configured. One method is to add an extension column to the base table, which would be populated using SVB code whenever an extension table was updated.

[Table 12](#) identifies the correspondences between system fields and system columns.

Table 12. System Fields and Their System Columns

System Field Name	System Column Name	Description
Id (or blank)	ROW_ID	Primary key for the table.
Created	CREATED	Creation date and time of the row.

Table 12. System Fields and Their System Columns

System Field Name	System Column Name	Description
Created By	CREATED_BY	Stores the ROW_ID of the S_USER record of the person who last updated the record.
Updated	LAST_UPD	Date of last update of the row.
Updated By	LAST_UPD_BY	Stores the ROW_ID of the S_USER record of the person who last updated the record. In some cases, this field is updated even though the user does not actively update the record. For example, this may occur when a multi-value link is configured with a primary join. See “Primary ID Field” on page 413 for more information.

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

Indexes

An Index object definition is the direct representation of a database index in a DBMS.

Siebel applications include a set of standard indexes. All indexes supplied by Siebel have the S_ prefix.

NOTE: You cannot modify or delete standard indexes.

You can create custom indexes if you determine that an additional index would benefit your implementation.

The Index object type has the following key properties:

- **Name.** The name of the database index.

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

NOTE: You can use Siebel Tools to create indexes in ascending or descending order. For Oracle, all indexes will be created in 'asc' mode (even if defined as 'desc' in Tools) since only the Rule Based Optimizer (RBO) is currently supported by the Siebel application for Oracle databases.

Index Column Object Type

Index Column is a child object type of the Index object. An Index Column object definition associates one column to the index that is the parent object definition of the index column. The Index Column object type has the following important properties:

- **Column Name.** The name of the column object definition to include in this index object definition.
- **Sequence.** The integer value that indicates the order of the column in the index relative to other columns, if more than one column is present.

- **Sort Order.** The sort order for the index column. Its value can be either Asc (ascending) or Desc (descending).

NOTE: Do not define indexes with names longer than 18 characters in the DB2 environments.

User Keys

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

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

A predefined index exists for each Siebel-defined user key. This index also takes the form *S_TABLE_NAME_Un*.

NOTE: Modifying user keys in standard Siebel tables or EIM base tables is a restricted activity and should not be attempted. Siebel Expert Services can assist customers in evaluating strategies to remap data in their implementations to make use of the current user key structure within their specific business requirements.

For more information, see [“EIM Interface Tables” on page 228](#).

EIM Interface Tables

Interface tables are intermediate database tables between the Siebel application database and other databases. A database administrator populates these tables with information to be processed by Siebel applications. You then invoke Siebel Enterprise Integration Manager (EIM) to process this information. EIM manages the exchange of data between Siebel database tables and other corporate databases. You can use EIM to perform bulk imports, exports, merges or deletes.

Interface tables have names that take the form EIM_.

All interface tables have a value of Interface in the Type property of their object definitions.

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

When tables or columns are added to the database, including extension tables and extension columns, it is generally necessary to configure the corresponding EIM and docking or routing interfaces. Siebel Experts typically perform the configuration of EIM and docking interfaces. Or you can use the EIM/Docking Wizard to extend Siebel EIM/Docking model. However, you can create and configure Attribute Mapping object definitions and view the other object types for the purpose of studying your existing interfaces. After you create an extension column, you must add one or more Attribute Mapping object definitions in interface tables that will supply data to it.

Two terms will be useful in the discussion that follows: *foreign key column* and *attribute column*. A *foreign key column* contains a reference to the primary key of another table, providing the means to perform a join from one table to the other. In contrast, an *attribute column* holds data, and does not point anywhere else.

NOTE: Do not modify the EIM attribute mappings shipped as part of the standard Siebel repository. Doing so may cause EIM to behave incorrectly.

EIM Object Types

EIM object types are illustrated in [Figure 56](#).

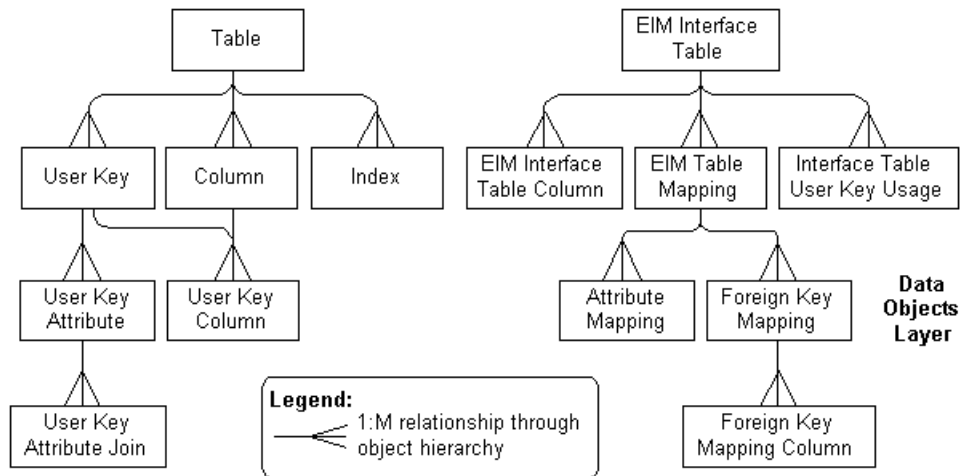


Figure 56. Architecture of EIM Object Types

The object types used in EIM configuration are described in the following section.

- **EIM Interface Table object type.** The EIM Interface Table object type is an alternative representation of the Table object type, for tables of type Interface only. That is, each interface table has a table object definition (with a value of Interface in the Type property) and an EIM interface table object definition. This is illustrated in [Figure 57](#).

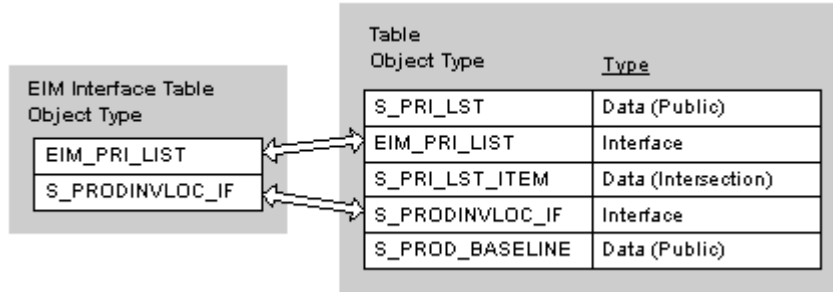


Figure 57. Relationship Between EIM Interface Table and Table

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

- EIM Interface Table Column object type.** The EIM Interface Table Column object type is an alternative representation of the Column object type, for columns that are child object definitions of interface tables. For a given interface table, the same list of columns appears as column children of the table object definition and as EIM interface table column children of the corresponding EIM interface table object definition. This is illustrated in [Figure 58](#).

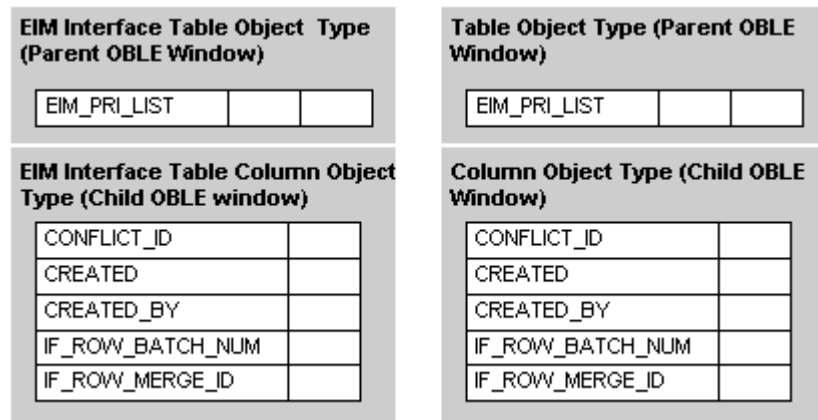


Figure 58. Relationship Between EIM Interface Table Column and Column

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

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

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

NOTE: Modifying user keys in standard Siebel tables or EIM base tables is a restricted activity and should not be attempted. Siebel Expert Services can assist customers in evaluating strategies to remap data in their implementations to make use of the current user key structure within their specific business requirements.

- **EIM Table Mapping object type.** Identifies a data table that is updated by the parent EIM interface table object definition. One interface table may update one or more data tables, and each data table to be updated requires an EIM Table Mapping child object definition of the EIM Interface Table object. Each EIM Table Mapping object definition identifies the name of the destination table (data table to update) in its Destination Table property. This is illustrated in [Figure 59](#).

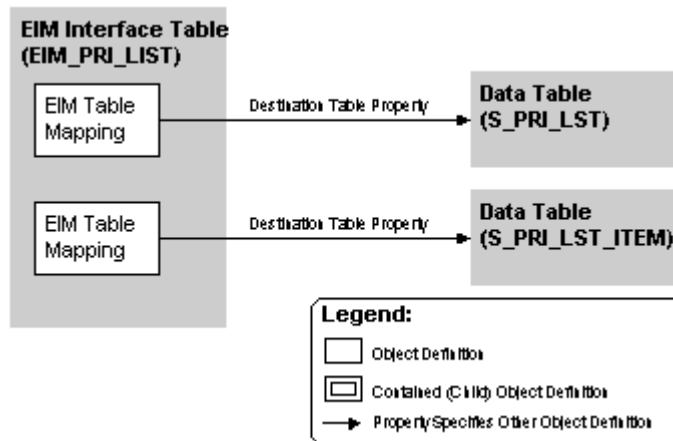


Figure 59. EIM Table Mapping Configuration

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

- Attribute Mapping object type.** Identifies an attribute (data) column to update in the destination (base) table specified in the parent EIM table mapping. Each Attribute Mapping object definition identifies the column in the interface table that supplies the data (in the Interface Table Data Column property). It also identifies the column in the destination table that receives the data (in the Base Table Attribute Column property). This is illustrated in [Figure 60](#).

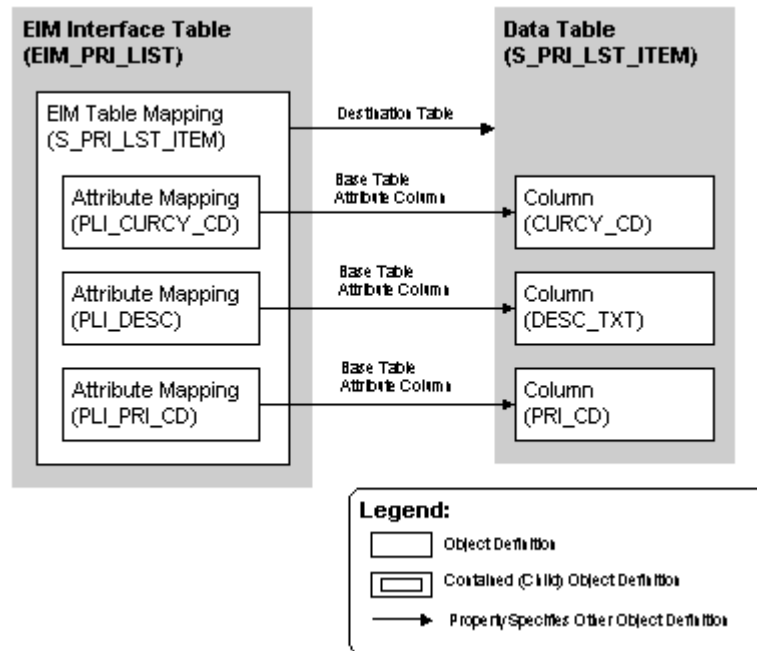


Figure 60. Attribute Mapping Configuration

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

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

A foreign key mapping is not a one-to-one column mapping from interface table to destination table, as occurs with non-foreign key columns. The numeric foreign key does not even exist in the interface table, so it cannot be mapped. Instead, a combination of attribute columns in the destination table of the foreign key is used to access the desired row, and the foreign key value can be obtained from that row. These relationships are illustrated in [Figure 61 on page 235](#).

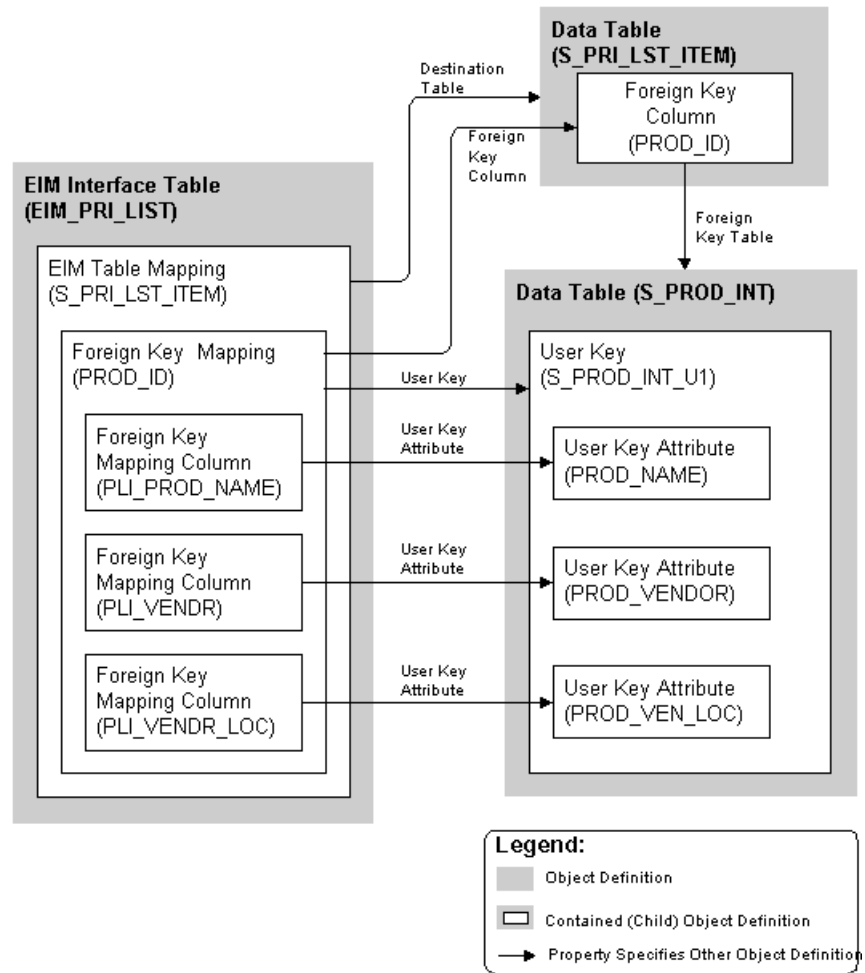


Figure 61. Foreign Key Mapping Configuration

- **Foreign Key Mapping Column object type.** Each Foreign Key Mapping Column object definition identifies a piece of the user key columns; that is, one of the attribute columns used to locate rows in the table the foreign key points to. The user key columns, taken together, uniquely identify rows in that table. The Foreign Key Mapping Column object definitions identify these user key columns to the interface table, so that foreign key values can be derived when import or export takes place.
- **User Key object type.** User Key is a child object type of Table. Each user key object definition provides a set of attribute columns and related information that specifies how the table's rows can be accessed in a particular EIM scenario. User Key has two child object types: User Key Column and User Key Attribute.
- **User Key Column object type.** User key columns can be either attributes or foreign keys. In most cases these are the columns in the user key index (usually the index with a suffix of _U1), with the exception of the CONFLICT_ID column.
- **User Key Attribute object type.** Each user key attribute object definition in the parent user key specifies one in the set of attribute columns that collectively identify rows in the grandparent table. The column name is specified in the Name property of the User Key Attribute object definition. User Key Attribute has one child object type, which is User Key Attribute Join.
- **User Key Attribute Join object type.** Each User Key Attribute object definition has one or more User Key Attribute Join child object definitions. The user key attribute join specifies a join operation that can be used to convert a user key attribute that is itself a foreign key to another table into attribute column values in that table. For example, the S_PROD_INT (products) table has a user key consisting of three attributes: PROD_NAME, PROD_VENDOR and PROD_VEN_LOC. The PROD_NAME (product name) attribute column is directly obtained from the S_PROD_INT table, so no join is required. However, the PROD_VENDOR and PROD_VEN_LOC columns occur in the S_ORG_EXT (accounts) table, and must be obtained using a join on VENDR_OU_ID, a foreign key from S_PROD_INT to S_ORG_EXT.

Adding and modifying attribute mappings are subject to the restrictions identified in [Table 13](#).

Table 13. Restrictions on Adding and Modifying Mappings

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

NOTE: Do not map multiple interface table columns to a single column in a target table. This creates ambiguity for EIM. However, you can map a single column in an interface table to multiple base tables or extension tables.

You can deactivate mappings if they are no longer necessary. To deactivate a mapping, navigate to the Attribute Mapping Object definition in the Object List Editor and place a check mark in the Inactive property. You should not delete any mappings.

NOTE: No validation is performed against interface table or column definitions. LOV validation is performed against the LOVs defined for the base columns to which they are mapped.

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

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

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

For more information, see [“Using the Check No Match Property with a Primary Join” on page 416](#).

Access Control

S_PARTY allows modeling of real-life relationships between different business entities. This section describes the role of S_PARTY; it allows you to configure business components related to access control and to import access control data and populate the necessary extension tables with data about persons and organizations.

S_PARTY and its extension tables are used to store data for many business components as shown in [Figure 62](#) below.

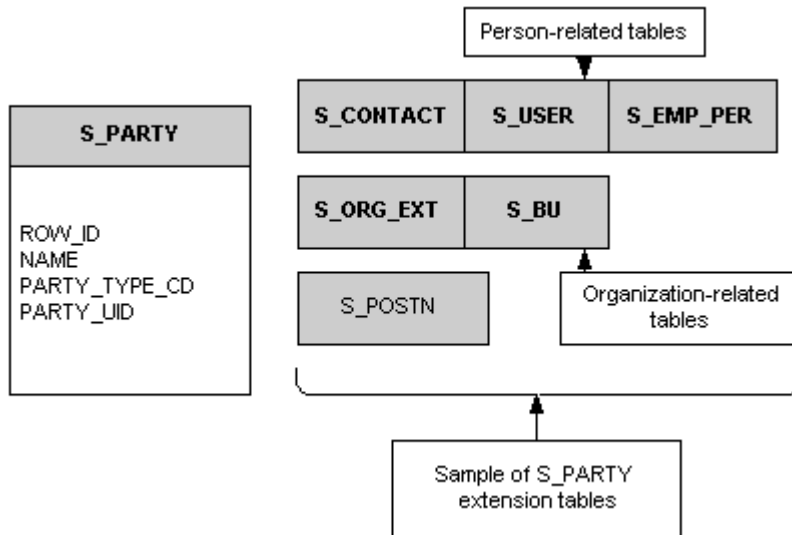


Figure 62. S_PARTY Overview

Party

Party refers to all types of Siebel person and business entities. This includes:

- Person-related entities (for example, Contact, Employee, User, Partner Employee)
- Organization-related entities (for example, Account, Position, Division, Organization, Household)
- Grouping for access control (for example, Access Group, User List). It also allows for grouping of instances of different types of entities.

Person-Related Data

A person refers to someone using the application or referred to in the application:

- Employee in a company that is using a Siebel application
- Individual at a channel partner
- Customer using the Web site
- Individual external to your company who is associated with the business process

Person-Related Business Components

Person-related business components store the majority of their data in S_CONTACT. Only the Siebel userID is stored in the S_USER table ([Figure 63 on page 241](#)).

Many business components also use these tables to store person-related data (Figure 63).

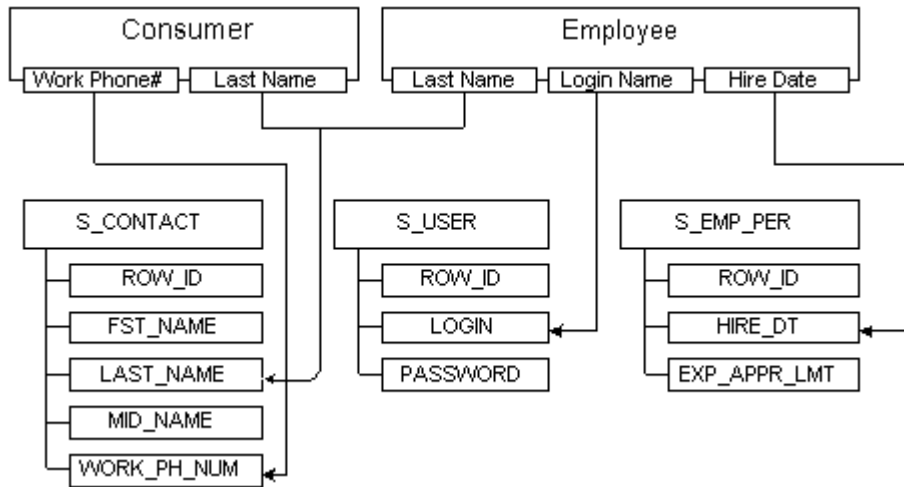


Figure 63. Many Business Components Store Person-Related Data

Relationships for Responsibility

User-Responsibility (many-to-many) relationships use the S_PER_RESP intersection table (Figure 64). Any user can be granted a responsibility.

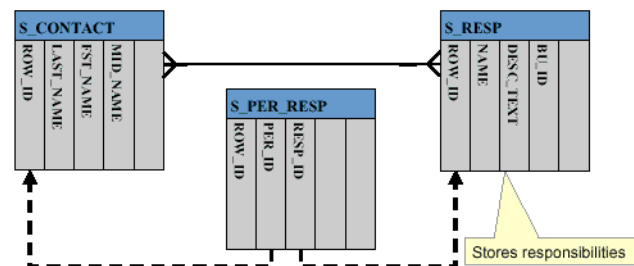


Figure 64. Relationships for Responsibility

Organization-Related Data

Organization-related data represents any business enterprise associated with a Siebel application:

- The company or part of the company using the Siebel application (division, organization)
- An external company that purchases your products (account)
- A partner company that assists you in your business (channel partner)
- A household

Organization-Related Business Components

Organization-related business components store their primary data in S_ORG_EXT and store additional data in S_BU (Figure 65).

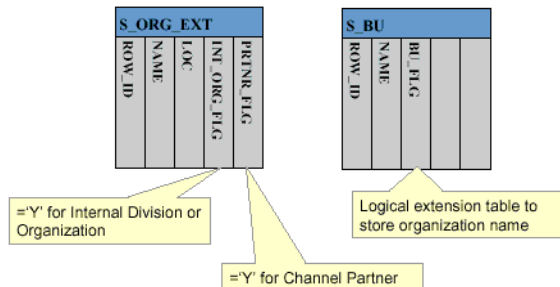


Figure 65. Organization-Related Business Components

Many business components use these tables to store organization-related data (Figure 66).

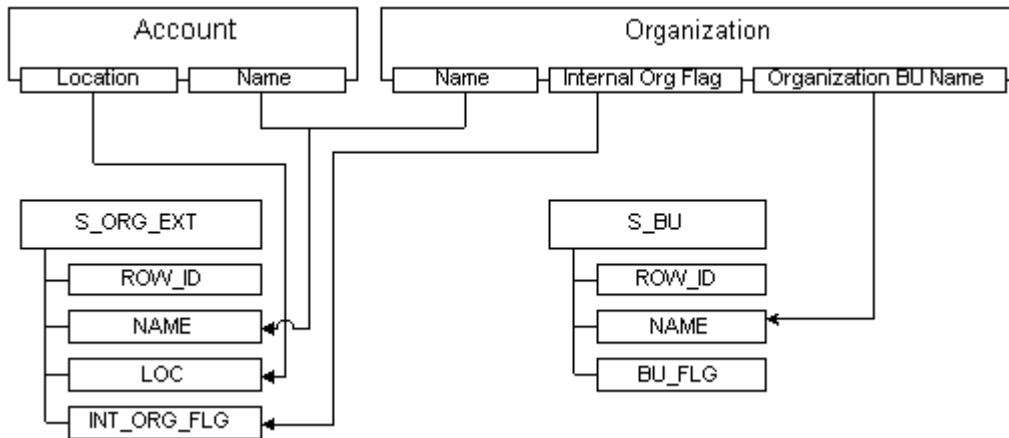


Figure 66. Many Business Components Store Organization-Related Data

S_BU Table

The S_BU table, shown in Figure 67, does the following:

- Permits indexing on organization name
- Supports organizational visibility

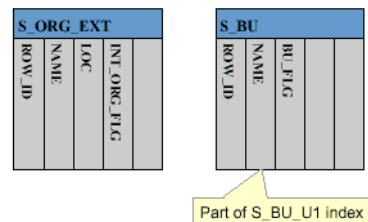


Figure 67. S_BU Table

Single-Organization Visibility

Single-organization visibility is implemented by the BU_ID foreign key column in the table for a single-organization business component ([Figure 68](#)).

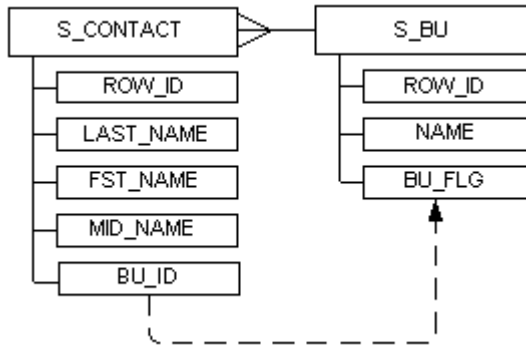


Figure 68. Single-Organization Visibility

Multi-Organization Visibility

Multi-organization visibility is implemented by an intersection table between S_BU and the table for the multiple-organization business component. Intersection tables for organizations have a _BU suffix (Figure 69).

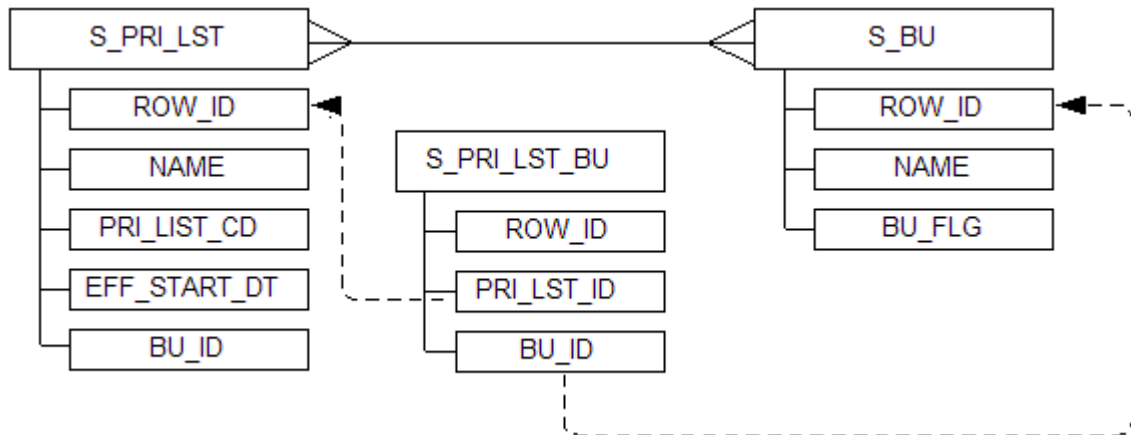


Figure 69. Multi-Organization Visibility

Party Business Components

Party business components consist of business components that represent people and organizational units. For example, the following list references the S_PARTY table.

- Account
- Contact
- User
- Organization
- Employee
- Position
- Household

S_PARTY Table

The S_PARTY table serves as the base table for all party business components and stores the party name and party type. It has multiple extension tables that store the business data for the party business components. The S_PARTY table allows you to model the complicated business relationships from the real world into Siebel applications (Figure 70).

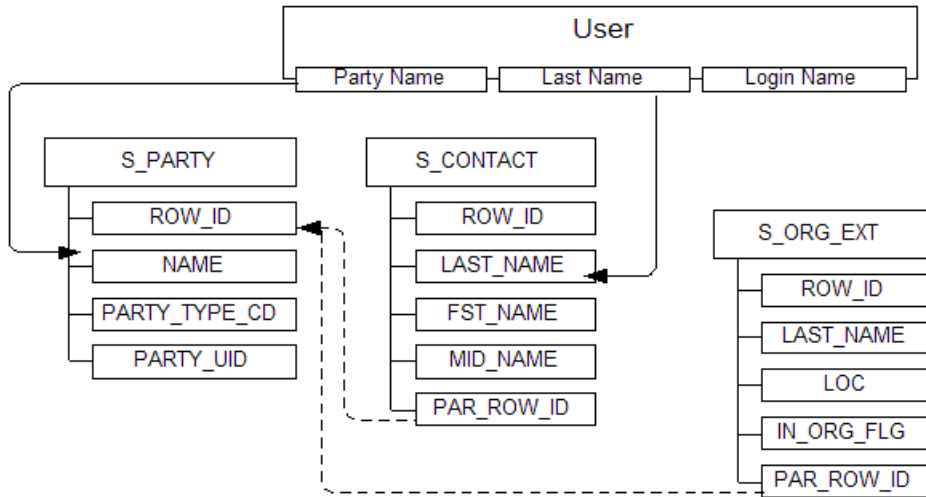


Figure 70. S_Party Table

Party

Party includes business components that represent groupings of party instances (Figure 71 on page 247).

- User List: Group of Users

- Access Group: Grouping of Access Group Members (for example, household, account, organization, division, position, user list).

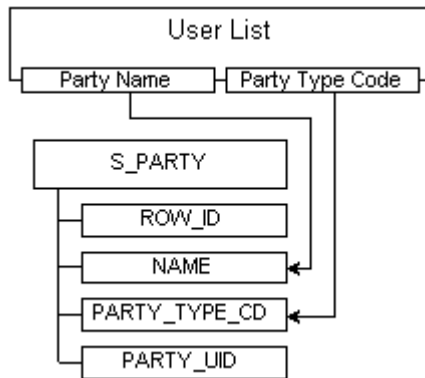


Figure 71. Party

S_PARTY_PER

S_PARTY_PER is an intersection table that relates two instances of parties and is used to implement group-member relationships between the following, respectively (Figure 72):

- User Lists and Users
- Accounts and Contacts
- Employees and Positions
- Access Groups and Members

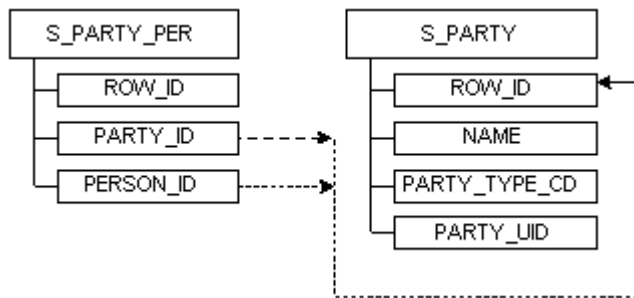


Figure 72. S_PARTY_PER

In the implied join between S_PARTY_PER and S_PARTY:

- S_PARTY_PER.PARTY_ID maps to the group's row ID (S_ORG_EXT.ROW_ID)
- S_PARTY_PER.PERSON_ID maps to the member's row ID (S_CONTACT.ROW_ID)

Summary

In summary, S_PARTY and its extension tables are used to store data for many business components (Figure 73).

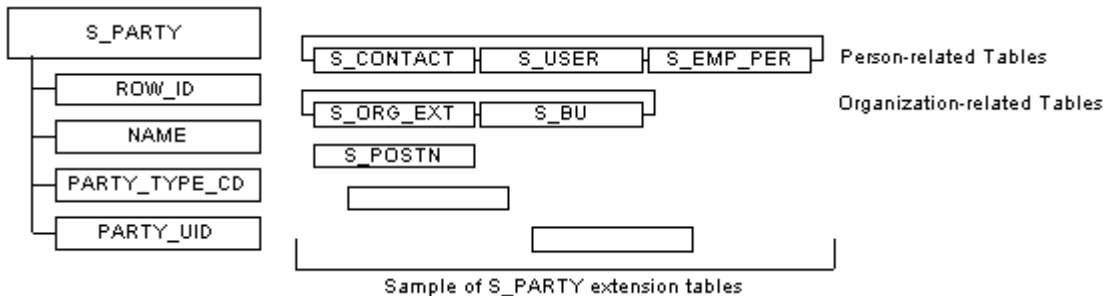


Figure 73. Overview of S_PARTY and Its Extension Tables

Use the following guidelines when working with S_PARTY-related tables:

- Configuring business components:
 - All person-related business components use S_CONTACT.
 - All organization-related business components use S_ORG_EXT.
 - Over 100 party-related business components reference S_PARTY but store their data in one of the many S_PARTY extension tables.
- Importing data for party-related business components:
 - You must populate columns in S_PARTY table in addition to tables that store the data of interest to users.

- Importing data for organization-related business components:
 - For single-organization data, you must populate BU_ID.
 - For multi-organization data, you must populate the corresponding intersection table.

Data Objects Layer

Access Control

This chapter describes the various options available for extending the Siebel Data Model.

About Extending the Siebel Data Model

Options for extending the Siebel data model include the following:

- **Standard (Static) Extensibility.** The standard extensibility includes the Siebel Database Extension Designer that allows you to add columns to Siebel tables or create 1:1 extension tables. For developers who require extensions beyond built-in database extensions, the Database Extension Designer provides a point-and-click interface for extending Siebel application tables. Customers can use these database extensions to capture data from new fields in application screens or from external sources using Siebel Enterprise Integration Manager (EIM).
- **Advanced Database Extensibility.** Advanced Database Extensibility includes the following wizards:
 - Table Wizard automates the generation of new table objects and standard system and foreign key columns. It allows you to create the following types of tables:
 - Stand-alone tables that are not related to any other type of table in the Siebel Data Model.
 - Tables that have a Many: One or One: One relationship with a table in the Siebel Data Model.
 - Tables intersecting between two existing tables in the Siebel Data Model.

- EIM Table Mapping Wizard allows you to create or associate the new table to the appropriate interface tables for using EIM. You can generate EIM Table Mapping objects for importing data into tables you have created, and you can automate the creation of EIM Attribute maps on extension columns added to base tables, which speeds up the development of complex EIM table mapping objects.
- The Dock Object Mapping Wizard allows you to associate the new table with an existing or new customer Dock object to support the synchronization of its data to remote users.

Benefits include:

- Ability to import data into custom tables using EIM
- Ability to synchronize the contents of custom tables to remote users
- Automatically generate basic index and User Key objects for new 1:1, M:1, and intersection tables.

NOTE: Advanced Database Extensibility should be used only after having exhausted other mechanisms including the use of prebuilt extensions and the Database Extension Designer. Changes made using Advanced Database Extensibility are upgradable from release to release and will not conflict with modifications made by Siebel.

- External Schema Import Wizard allows you to create a new Table object in the siebel repository based upon the contents of a DDL. See [“Advanced Database Extensibility” on page 292](#) for further information.

About Standard Database Extensibility

Standard Database Extensibility refers to capabilities in Siebel Tools allowing you to extend the Siebel eBusiness Applications Data Model. It has two major components:

- **Static Database Extensions.** The Siebel eBusiness Applications Data Model contains several columns and tables that you can customize for your own purposes. These columns and tables are already part of the Data Model and require no schema modifications.

If you need to track information that is not captured in the base entities, your first option is to determine if the static extensions can be used to solve the problem.

The following are the types of static database extensions:

- **Static Database Column Extensions.** The Siebel Data Model includes over 30,000 columns in Base Tables that are specifically included for you to use for your own purposes.
- **Static 1:1 Database Table Extensions.** The Siebel Data Model includes over 40 tables that provide 1:1 relationships with Base tables that are specifically included for you to use for your own purposes.
- **Static Many:1 Database Table Extensions.** The Siebel Data Model includes over 20 tables that provide 1:Many relationships with Base Tables that are specifically included for you to use for your own purposes.
- **Database Extension Designer (Dynamic Database Extensions).** For developers who require extensions beyond built-in database extensions, Siebel Database Extension Designer provides a point-and-click interface for extending Siebel application tables. Customers can use these database extensions to capture data from new fields in application screens or from external sources using Siebel Enterprise Integration Manager (EIM). Siebel Database offers the following extension capabilities:
 - **Dynamic Database Column Extensions.** You may add your own columns to any existing Base Table in the Siebel Data Model.
 - **Dynamic Database Table Extensions.** You may add your own tables bearing 1:1 relationships with Base Tables in the Siebel Data Model.

It allows you to do the following:

- **Capture additional attributes on entities.** You can add columns to database tables. You can use these columns to store additional information for use by business object definitions. You can then modify the user interface object definitions to display and update the contents of added columns.

- **Create your own extension tables.** You can also extend the Siebel Data Model by creating new 1:1 extension tables for Siebel base tables.
- **Carry custom attributes forward with new releases.** As part of the implementation process, the Database Extension Designer generates the necessary database-specific DDL commands, and registers the columns in the Siebel application dictionary. Siebel applications recognize the columns, and carries them forward into subsequent releases of the application.
- **Integrate with Siebel Remote.** After implementing a database extension, you can request that Siebel applications automatically generate a new structure for mobile client local databases. Siebel applications provide a standard methodology for propagating changes to mobile clients.

The Database Extension Designer uses a point-and-click interface for adding columns to tables. It works in conjunction with the Object Explorer and Object List Editor to allow you to view all tables in the database and all columns in each table, and add extension columns to a table as necessary. This approach simplifies the process of implementing necessary changes, allowing you to meet the needs of your users more quickly.

NOTE: You need to have a thorough understanding of the Siebel Object Architecture and Siebel Data Model before you undertake database extension.

Using Standard Database Extensibility

You can use Standard Database Extensibility to implement database extensions that meet various business requirements, without having to resort to Advanced Database Extensibility. For example:

■ Many: One Tables

Requirement: You need to track an entity that does not exist in the Siebel eBusiness Applications Data Model and that bears a Many: One relationship with a Base Table.

Case 1: The Base Table already has an `_XM` table associated with it.

In this case, you can use this table to track multiple entities by using the `TYPE` column to group records.

Case 2: The Base Table does not have an `_XM` table associated with it, but there is another table in the Data Model that bears a Many: One relationship to the Base Table and is not being used in the Siebel implementation.

In this case, you may repurpose this table for your needs by adding columns to it to track the necessary attributes.

■ Intersection Table

Requirement: You need to implement a Many: Many relationship between two tables (T1 and T2).

- Look for an existing `_XM` table for T1 or another table that bears a Many: One relationship to T1 and is not used in the implementation. If such a table exists, you may be able to use it as an Intersection table.
- If the table identified in step a is an `_XM` table, use the `NAME` column as the Foreign Key to the table (T2). You will also need to make the configuration changes necessary to make sure that the `TYPE` column gets populated with a default value.
- If the table identified in step a is an unused table, then you will need to choose an appropriate (required) column that can serve as a Foreign Key to the table (T2). You will also need to make the configuration changes necessary to make sure that all other required columns are populated.

- Stand-Alone Table

Requirement: You need to create a table that is not related to any other tables in the data model; this table does not have to be available through synchronization.

Case 1: The data exists in an external system or database.

You should consider using a Virtual Business Component to access this table. This option has the added advantage that there is no need to replicate the data within the Siebel Data Model. For more information about Virtual Business Components, see *Integration Platform Technologies: Siebel eBusiness Application Integration Volume II*.

Case 2: The data does not exist in an external system or database.

You could create the table, possibly in the same database and under the same table owner as the rest of the Siebel Data Model. Though this table is not represented in the Repository, its data can be accessed through the Virtual Business Component mechanism.

Database Extension Planning and Design

Siebel Tools and Siebel applications provide several tools that automate much of the process of implementing database extensions. However, to be successful, you need to plan and design your extensions before using these tools. Planning and design tasks are listed in this section, and those that the Database Extension Designer facilitates are described in more detail in subsequent sections.

You should use a test database environment where you can make and verify database extensions before they are propagated to the production database.

NOTE: Be sure to lock these projects when extending the database through Siebel Tools: Newtable, EIM Interface Table, and ERP Interface Table.

Planning and Design Steps

The following procedures describe the steps for planning and designing database extensions.

To plan for adding a custom column to the database

- 1** Decide whether a new column is needed, or whether a standard extension column such as ATTRIB_03 or ATTRIB_27 in the standard extension table will meet your needs.

In many cases it is better to use a new column in the base table rather than a standard extension column, because it gives somewhat better performance (that is, you avoid a join to the extension table). However, there are some exceptions to this guideline. Siebel Expert Services can identify when these exceptions might apply during a review such as a configuration design review or EIM mapping review.

If you need to store additional data that almost always exists for a given base record and is regularly accessed, the recommended approach is to extend the base table to store this data. By doing this you avoid an extra join to an extension table. However, this might result in slower access to the base table if there is a lot of data (that is, many large fields have been added and they are always populated), because fewer rows now fit on one page.

If a large number of extension fields is required, and if the view displaying this data is accessed infrequently, use an extension table. A join is executed for the extension table, but only when this view is accessed.

NOTE: When you create a new extension column in the Siebel schema, there might be padding issues with Siebel Remote. For details regarding what data type to use, see “Generating a New Database Template” in *Siebel Remote and Replication Manager Administration Guide*.

- 2** If you choose to use a standard extension column, verify that the column is not already in use by a field.

To find any fields currently using the standard extension column, do the following:

- a** In the Siebel Tools Object Explorer, click the Flat tab.
- b** Select the Field object type.

- c** Initiate a query by choosing Query > New Query from the menu bar. In the QBE row in the Object List Editor, enter the name of the column you want to use (in the Column property) and the name of the extension table (in the Join property). Press ENTER.
- d** If the query does not return any Field object definitions, the column is unused in the extension table and is available. If the query returns one or more object definitions, find another extension column in that table. To determine which extension columns are currently in use, perform the query again with the same extension table specified (in the Join property) and the value "ATTRIB*" in the Column property.

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

- 3** If you will add a new custom column, decide whether you will be implementing it after you have deployed active mobile clients. Mobile clients will either have to be reextracted to get the new column, or Siebel Anywhere will need to be used to distribute the schema change.
- 4** Design the extension column. Decide on a description, name, user name, and characteristics (such as data type, length, and default values).
- 5** Decide on the applet, view, and screen where the column will be used. Decide if users can only view column data, or view and update it.
- 6** Decide where your custom column will reside: in a base table, a standard extension table, or a custom extension table.

To find any existing columns in which you might place the field, do the following:

- a** In Siebel Tools, navigate to the applet to which you want to add a control or list column.
- b** Note the value in the Business Component property.

- c** Navigate to this business component using the Object Explorer and Object List Editor. The object definition for the business component in the Object List Editor displays, in its Table property, the name of the base table. Note the name of this base table.
- d** In the Object Explorer, select the Table object type.
- e** In the Object List Editor, perform a query to restrict the list of tables to just display extension tables with a base table name matching the previously identified base table, as follows:
 - Type property set to a value of Extension
 - Base Table property equal to the base table's name, in all capitals

The result of this query is a display of the extension tables for the identified base table.

NOTE: Alternatively, you could review the base table to determine if it would be appropriate to create an extension column there. This would require a different Object List Editor query, in which you set the Name property to the Name of the identified base table, and expand the Column object type in the Object Explorer.

- 7** Decide on the method for loading data into the extension column. You may be able to populate the extension column using the Enterprise Integration Manager (EIM). For more information, see [“Populating Extension Tables and Columns” on page 290](#). Alternatively, you could have your users enter data through the user interface, or use Siebel Object Interfaces and Siebel VB or Siebel eScript to update the data programmatically.
- 8** Document the planned changes, and work with Siebel Technical Services to review the design.

- 9 Develop an implementation schedule that includes adequate time to test and implement your work, as well as to propagate the changes to your active mobile clients.

NOTE: Because all extensions are made and tested first against a local development database, then made and tested against the test server environment, your mobile users can continue to synchronize as you implement database extensions.

Naming Conventions for Extension Tables and Columns

The Database Extension Designer enforces a naming convention for extension tables and columns to prevent naming conflicts and make sure that you can upgrade your application.

Extension Table Names

The table name for an extension table comprises the name of the base table, an underscore, and a suffix. The “X” suffix denotes a one-to-one (1:1) extension table; the suffix “XM” denotes a one-to-many (1:M) extension table. For example:

- S_OPTY_X is the name of the 1:1 extension table for the S_OPTY base table.
- S_OPTY_XM is name of the 1:M extension table for the S_OPTY base table.

NOTE: Custom extension tables have a prefix of CX_. This prefix is used to distinguish custom tables that have been created from tables provided by your Siebel application. Prefix CX_ is used to distinguish custom tables from *all* Siebel tables, not just Siebel Extension tables.

The user name of an extension table (in the User Name property) comprises the user name of the base table, a space, and a suffix. The suffix Extension denotes a 1:1 extension table; the suffixes M:1 Extension or 1:M Extension denote a many-to-one or one-to-many extension table. For example:

- Opportunity Extension is the user name for the S_OPTY_X extension table.
- Opportunity M:1 Extension is the user name for the S_OPTY_XM extension table.

Extension Column Names

The column name for an extension column in a base table consists of a prefix, an underscore, and a column name. For example:

- X_DISCOUNT
- X_SPECIALCHG

The user column name for an extension column in a base table uses a column name and a suffix. For example:

- Discount Extension
- Special Charge Extension

NOTE: The Database Extension Designer imposes no naming convention on extension columns in an extension table. Be careful not to use database reserved words.

There are eight predefined system columns added to a Customized Extension table you create. However, as part of the standard Siebel application, there are 47 predefined data columns of varying types in standard extension tables.

Accommodating Active Mobile Clients

Whenever possible, you should implement custom attributes before extracting mobile clients. If you can do so, then propagating custom attributes to these clients occurs during normal client installation.

DBMS Restrictions

You can add extension columns to a base table, standard extension table, or custom extension table.

However, you should be aware that different DBMS systems have different limits on the width of a table row, either on the maximum number of columns in a table or the maximum number of bytes in a row. If you need to exceed the limit, you can add extension columns to custom extension tables only. Some examples of DBMS systems limitations are:

- Microsoft SQL Server 7.0 has a maximum Varchar length of 8000.
- Customers cannot add long columns to Siebel tables. However, customers can add one long column to custom extension tables in Oracle.

After it has been added to the physical table, a column cannot be removed. Refer to your database system's documentation for information on this constraint.

NOTE: Customers are advised not to decrease the length of a column because this can cause data loss [for example, decreasing a varchar (30) to varchar (20)] or to decrease the precision [for example, decreasing from number (10,3) to number (10,1)] because this can cause a loss of precision.

Database Extension Implementation

After planning and designing your database extensions, complete the following steps to implement your extension tables and columns on the Siebel test database server. If you are performing this work after your Siebel applications are in production, you should establish a separate test environment and database. This approach assures the safety and integrity of your production system as you develop and test your database extensions.

NOTE: During the process of implementing extension columns and tables, Siebel Tools generates and executes the SQL data definition language (DDL) that is used to modify the database and update other Siebel system components.

As with configuration work, database extensibility must first be performed against the local database. This allows you to recover the data in the event of mistakes, and allows you to test thoroughly the changes before making them available to all developers.

CAUTION: Creating tables using SQL is not supported; use only Siebel Tools to create tables and perform other logical database schema extensions.

To implement database extensions

- 1** Check out and lock the project to which the table being extended belongs.
- 2** Update the logical schema definition in the local environment.
- 3** Apply the physical schema extensions to the local database.
- 4** Update and test configuration changes that apply to the extension.
- 5** Prepare the server database prior to applying schema extensions.
- 6** Apply the changes to the server database and click Activate.

The Activate button makes the changes effective for server tasks like EIM and Generate New Database.

- 7** Apply the server database changes to other local databases.

The following sections discuss each step in detail. Once you have implemented your extensions in your test environment, refer to the section entitled “[Migrating Repositories and Schemas Between Databases](#)” in [Chapter 17, “Repositories](#)” for instructions on implementing these extensions in your production database.

Checking Out and Locking the Projects

With Siebel Tools running against the local database, check out the project to which the table you are extending belongs (in most cases this is Newtable). Be certain to specify Server Lock when checking out the projects.

NOTE: If you want to import data into the new column using EIM, you need to check out whatever project the appropriate interface table belongs to.

Updating the Logical Schema Definition in the Local Environment

The logical schema (data object definitions) must be updated with your extensions in Siebel Tools before you apply them to the physical database. Logical schema update is performed against your local development repository. Any changes made to your logical schema will be indicated by a check mark in the Changed column for the table or column affected.

This section is organized into the following subsections:

- [“Creating a Custom Extension Table” on page 264](#)
- [“Adding Extension Columns to Tables” on page 266](#)
- [“Modifying Extension Tables or Columns” on page 269](#)
- [“Deleting Extension Tables or Columns” on page 270](#)
- [“Using Extensions with Enterprise Integration Manager” on page 272](#)
- [“Adding Custom Indexes” on page 274](#)

Carry out all of the steps in all of the subsections, in the order given here.

Creating a Custom Extension Table

Siebel eBusiness Applications provide standard extension tables for the major base tables in the schema. Each extension table has standard extension columns that you can use to hold custom data. In some cases, however, you may need to create custom extension tables.

When configuring custom extension tables, consider the following:

- If you wish to create an extension table for tables whose type is Extension (Siebel), you must extend from S_PARTY. For example, S_CONTACT is an extension table of S_PARTY. Because S_CONTACT is of type Extension (Siebel), you cannot use it as a parent table for an extension table. You must use S_PARTY. For a business component based on your new table to show data from S_CONTACT, you must create a Join object that references S_CONTACT and has a Join Specification child object with a Source Field property set to Parent Id and Destination Column property set to ROW_ID. The row ID of an S_CONTACT record will be the same as the row ID of the corresponding S_PARTY record. For more information about S_PARTY, see [“Access Control” on page 238](#) for more information.
- After creating a new table using the Extend button, you will need to restart Siebel Tools if you want to delete any custom tables.
- Custom 1:1 extension tables do not need new docking rules, as the data contained in these tables is implicitly routed according to the docking rules of their parent tables.
- Limit the number of extension tables for a given base table to help avoid performance problems. The number of extension tables is also limited by the underlying database capacity.
- You cannot add more than 20 extension tables to a base table. If for some reason you need more than 20 extension tables on a base table, you must use the Advanced Database Extensibility option. For more information, see [“Advanced Database Extensibility” on page 292](#). You should also consult Siebel Expert Services.

To create a custom 1:1 extension table

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

4 Click Extend.

The extension table appears in the list of tables in the Object List Editor. The Database Extension Designer will automatically create the necessary standard columns and standard indexes. Temporary columns in interface tables that are imported to the base table for this extension table are also created automatically.

5 Create any additional extension columns on the custom extension table, following the instructions in [“Adding Extension Columns to Tables.”](#)

Adding Extension Columns to Tables

You can add extension columns to any of the following table types:

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

You should check the Type property for a table to verify that you can extend it:

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

NOTE: For Oracle databases, the maximum length of extension columns with data type of Varchar is 2000 characters. If you create a column of type Varchar that is longer than 2000, it is implemented as a column with data type of LONG.

To add an extension column

- 1 Connect to your local development database with Siebel Tools.
- 2 In the Object Explorer, expand the Table object type.

- 3 In the Object List Editor, select the table to which you will add an extension column or columns.
- 4 Make sure that the table is not of type Data (Private).
- 5 Select Column in the Object Explorer.
- 6 Select Edit > New Record to add an extension column.

NOTE: Do not use column names that are reserved words on your server or client database.

When you add columns to base tables or interface tables, Siebel Tools enforces the naming conventions described in the section [“Naming Conventions for Extension Tables and Columns”](#) on page 260.

You must provide a default value, in the Default property, for any column that you designate as mandatory (with a FALSE value in the Nullable property). Although you can create a mandatory column without providing a default, you will encounter problems when using this column. For example, if you try to add a column to a table that is already populated with data, the database will fail to create the column. If the table does not yet contain data, attempts to add data (through the user interface, Enterprise Integration Manager, or Siebel Remote) may fail.

In an extension column, use only default values for the following properties:

- Translate
- Use FKey
- Sequence Object
- Cascade Clear

- Foreign Key Table Name

CAUTION: Be extremely careful when using custom extension columns to track foreign keys. If you choose to implement this, you must consult with Siebel Expert Services concerning the visibility rules applied to the foreign key table. Additionally, you must set the Foreign Key Table Name property to NULL for that column to load values into it using Enterprise Integration Manager.

For information on creating foreign key mappings for EIM, see [“EIM Table Mapping Wizard” on page 299](#).

Creating Extension Columns of Type LONG

You can create an extension column of type LONG, but you are limited to one column of this type per table. When configuring LONG columns, consider the following:

- You can only add LONG columns to 1:1 Extension Tables that have a valid base table identified in the Base Table property of the table object definition.
- You cannot add LONG columns to tables of type Data (Public), such as S_EVT_ACT. Siebel Systems reserves the right to implement LONG columns in these tables in the future.
- You can use LONG columns store a maximum of 16 KB or 16383 characters.

To create a LONG extension column

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

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

- 2** For the column that is created in the table, set the Physical Type to 'Long' and Length set to '0'.

You apply the LONG extension column like any other custom extension column.

- 3** Create a new field that maps to the extension column.

Modifying Extension Tables or Columns

You may need to modify your extension tables or columns after creating or activating them. You can modify properties of extension tables or columns only. Modification of standard base tables and their columns is not supported. If you delete the custom table, you need to delete the data object as well, if there is any.

To modify an existing extension column

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

NOTE: If you are adding a new extension table to the EIM Table Mapping list, make sure you click Activate to create all the necessary temporary columns that are needed for EIM processing.

- 4 In the Object Explorer, select Column.
- 5 In the Object List Editor, select the extension column to modify.
- 6 Modify the appropriate properties.

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

To rename an existing extension column

- 1 Deactivate the unwanted column.
- 2 Create the new column in Siebel Tools (the logical schema).
- 3 Export the data from the old column.
- 4 Drop the old column.
- 5 Use `ddlsync.ksh` to synchronize the logical and physical schema and import the data back in.

To modify an existing extension table

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

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

Deleting Extension Tables or Columns

If you have created an extension table or column that is no longer required, you can delete it from the logical schema. You can delete only custom extension columns and tables; you cannot delete any standard tables or their columns.

To delete an existing extension column

- 1 Connect to your local development database with Siebel Tools.
- 2 In the Object Explorer, expand the Table object type.
- 3 In the Object List Editor, select the table containing the extension column to be deleted.
- 4 In the Object Explorer, select Column.
- 5 In the Object List Editor, select the extension column to be deleted.
- 6 Select Edit > Delete from the menu to delete the extension column.

NOTE: Siebel Tools does not cascade the deletion of an extension column. You should delete or inactivate the attribute mapping after deleting the extension column.

To delete a mapping, navigate to the Attribute Mapping object in the Object List Editor and select Edit > Delete Record. To deactivate a mapping, set the Inactive property to TRUE for the Attribute Mapping object definition.

To delete an existing extension table

- 1** Connect to your local development database with Siebel Tools.
- 2** In the Object Explorer, select the Table object type.
- 3** In the Object List Editor, select the extension table to delete.
- 4** Select Edit > Delete from the menu to delete the extension table.

Extension tables or columns that are deleted are removed from the logical schema. If you delete an extension table, the corresponding temporary columns in any interface table that is imported to the base table for the extension table are not deleted. These columns cannot be deleted through Siebel Tools, and will remain in the logical and physical schema.

After you have deleted an extension table or column from the logical schema in the Siebel repository, the table or column still exists in the physical schema of the database. You must apply your changes to the physical database by clicking the Apply button. This will synchronize the logical and physical schemes, dropping any tables or columns from the database that you have deleted from the repository. See [“Applying the Physical Schema Extensions to the Local Database” on page 275](#) for more information.

NOTE: In some database platforms, such as DB2, dropping a column requires you to reconstruct the entire table. However, if there is an object based on the table, a DB2 object such as a view for example, you cannot drop the table from the database.

Using Extensions with Enterprise Integration Manager

If you plan to use the Enterprise Integration Manager (EIM) to populate an extension column with data, you could manually add an extension column or table to the appropriate interface table, and map this extension column to the extension column in the base table or extension table. Or you could use the EIM Table Mapping Wizard to generate mapping automatically. See “[EIM Table Mapping Wizard](#)” on page 299.

NOTE: After the EIM objects are modified, you do not need to compile and distribute a new .srf with these changes for EIM to recognize the new mappings. They are read directly from the database.

The Database Extension Designer and EIM support mappings between the columns in extension tables and columns in interface tables. When you create an extension table, you need to extend the EIM table manually and map the new columns.

You can make new mappings from an interface table to a base table if either of two conditions is true:

- There are already mappings from the interface table into the base table.
- The target table is an extension table, and there are already mappings from the interface table into the corresponding base table.

NOTE: Interface table names start with EIM_ in Siebel 7.5.

For example, you can create an extension column in the EIM_ACCOUNT table called X_CUST_NUM, and map this either to an extension column you added to S_ORG_EXT or to an existing column in the S_ORG_EXT_X extension table.

NOTE: Mappings from interface extension columns to base columns are not supported.

The following EIM Table Mapping properties are to be used only by Siebel Systems and should not be modified by customers:

- EIM Duplicate Proc Column
- EIM Exists Proc Column
- EIM ROW_ID Proc Column
- EIM Status Proc Column
- EIM Status Proc Column

To create and map interface extension columns

- 1** Identify the interface table in the Table Object List Editor that is used to populate your extension column.
- 2** Create the extension column on the interface table, following the instructions in [“Adding Extension Columns to Tables” on page 266](#).
- 3** In the Object Explorer, select the EIM Interface Table object type. In the Object List Editor, select the appropriate interface table.
- 4** In the Object Explorer, select the EIM Table Mapping object type. In the Object List Editor, select the base table or extension table that contains the column to be populated.
- 5** In the Object Explorer, select the Attribute Mapping object type.
- 6** Select Edit > New Record from the menu bar to create a mapping from the interface extension column to the appropriate extension column in the base table.

NOTE: Verify that the new interface extension column's Inactive flag is not checked. The new interface extension column will appear in the drop-down list whether the column's Inactive flag is checked or not.

EIM Processing Columns do not show in the drop-down list. Therefore if the new interface column does not show in the drop-down list, change the column's EIM Processing Column Flag from TRUE to FALSE.

You should not map multiple interface table columns to a single column in a target table. You can, however, map a single column in an interface table to multiple base tables or extension tables for a base table of the interface table.

You can either delete or deactivate mappings if they are no longer necessary. To delete the mapping, navigate to the Attribute Mapping object in the Object List Editor and select Edit > Delete Record. To deactivate a mapping, set the Inactive property to TRUE for the Attribute Mapping object definition.

Adding Custom Indexes

You can create custom indexes to enhance the performance of your implementation.

NOTE: You cannot modify standard indexes, or delete them from the schema.

When implementing custom indexes, consider the following:

- While adding an index may improve the performance of one query, it might adversely affect the performance of others. It is strongly recommended that you consult Siebel Expert Services to help evaluate the potential impact of a custom index in your database.
- If at some point you no longer require a custom index that you have created, do not delete it from the Siebel repository. Instead, inactivate it by selecting the object's Inactive property check box in the Object List Editor.

Deleting the custom index from the repository would remove the record from the logical schema but not from the physical schema. This would result in the logical and physical schemas being out of sync.

- Any custom indexes should be thoroughly tested in a test environment before being introduced into production.

To create a custom index

- 1 Connect to your local development database with Siebel Tools.
- 2 In the Object Explorer, expand the Table object type.
- 3 In the Object List Editor, select the table to which you want to add an index.
- 4 In the Object Explorer, expand the Index object type.

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

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

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

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

- 6 In the Object Explorer, select the Index Column object type.
- 7 Select Edit > New Record to specify each column to add to the index.

Applying the Physical Schema Extensions to the Local Database

Once your changes are complete, you are ready to update your local environment.

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

To update your local environment

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

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

The Apply Schema dialog box appears.

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

NOTE: When applying changes to an Oracle database, the privileged user id and password must be the tableowner name.

Field	Description
Tables	Select one of the following options from the drop-down menu: <ul style="list-style-type: none">■ All. Update the database to reflect all changes made to the dictionary. This option forces each database object to be compared with the data dictionary, and updated if required.■ Current Query. Update the database to reflect modifications made to the tables in the current query only.■ Current Row. Update the database to reflect modifications made to the table in the current row only.
Table space	Leave blank.
16K table space	Leave blank.
32K table space	Leave blank.
Index space	Leave blank.
Table groupings file	Optional. This file is provided by the DBA and is specific to your database.
Privileged user id	Enter your database user ID, for example SADMIN. The table owner is read from <code>tools.cfg</code> .

Field	Description
Privileged user password	Enter your database user password, for example SADMIN. Note: When the database initialization for a mobile client is performed, the table owner changes from SIEBEL to the mobile user's password. In this case, use the mobile user's password in the password field.
ODBC data source	Verify that the ODBC connection specified in the ODBC Data Source text box is correct for your environment. You cannot apply schema changes to any database other than the one you are currently connected to (for example, by specifying the ODBC name of a different database).

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

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

To make your new database extensions available to other developers

- 1** Have the developers check out the relevant projects from the server. This gives them the schema change logically, that is, in Siebel Tools object definitions.
- 2** Have them apply these changes to their local databases to obtain the changes physically.

Displaying Extension Data

Typically, data model changes are exposed in applets. You should test all of the new extension tables or columns, checking out updated copies modified by other developers, if necessary, against the local development database.

NOTE: When compiling the new .srf file, compile the projects.

Displaying Base Table Extension Column Data

Configuring an applet to display data from an extension column of a base table involves creating or modifying fields in the business component on which the applet is based, and then exposing these fields as controls or list columns.

To display data from an extension column of a base table

- 1** In the Object Explorer, expand the Business Component object type.
- 2** In the Object List Editor, select the business component to which you will add a field.
- 3** In the Object Explorer, select the Field object type.
- 4** In the Object List Editor, select Edit > New Record.
- 5** In the Column property in the new Field object definition, specify the column in the business component's base table that the field will represent.
- 6** In the Object Explorer, select the Applet object type.
- 7** Use the Applet Designer to add a control or list column in which this field will be displayed.
- 8** In the new control or list column object definition, specify the name of the new field.

Displaying Data in One-to-One Extension Tables

Configuring an Applet to display data from an extension column of a 1:1 extension table involves adding a field based on a join to the business component.

To display data from a 1:1 extension table

- 1** In the Object Explorer, expand the Business Component object type.
- 2** In the Object List Editor, select the business component to which you will add a field.
- 3** In the Object Explorer, select the Field object type.
- 4** In the Object List Editor, select Edit > New Record.
- 5** In the Column property, specify the column from which you wish to map this field.
- 6** In the Object Explorer, select the Applet object type.
- 7** Use the Applet Designer to add a control or list column where this field will be displayed.
- 8** In the new Control or List Column object definition, specify the name of the new field.

Displaying Data from One-to-Many Extension Tables

Displaying data from an extension column of a 1:M extension table involves creating a new business component and fields for the 1:M extension table, and creating a link and business object component to provide a master-detail relationship between the new (detail) business component and its master business component. Create an applet to represent the new business component, and a new view to display the link relationship.

To display data from a 1:M extension table

- 1** In the Object Explorer, expand the Business Component object type.
- 2** In the Object List Editor, select Edit > New Record.
- 3** In the Table property, specify the name of the 1:M extension table.

- 4 Specify all other necessary properties for a business component.
- 5 In the Object Explorer, select the Field object type.
- 6 Create a link between the existing master and the new (extension table-based) business components.
- 7 Add the new business component to the appropriate business object (by adding a business object component child of the business object), and specify the link with the master business component.
- 8 Create an applet that displays the detail business component. Create and administer a master-detail view using the new applet (the one that displays the detail business component).

Preparing the Server Database Before Applying Schema Extensions

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

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

NOTE: If you are making schema changes to custom extension columns and tables that already have data in the production environment (server database), you must export that data before making the schema changes. After making the changes, import the data back into the production environment.

Applying the Changes to the Server Database

Check the projects back in to the server database to update the repository schema definition there.

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.

Applying Schema Changes Using Siebel Tools

Use the Apply dialog to apply schema changes to the server database.

To apply and activate the changes to the server database

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

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

- 4** Click OK.

The Apply Schema dialog box appears.

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

Field	Description
Tables	Select one of the following options from the drop-down menu: <ul style="list-style-type: none">■ All. Update the database to reflect all changes made to the dictionary. This option forces each database object to be compared with the data dictionary, and updated if required.■ Current Query. Update the database to reflect modifications made to the tables in the current query only.■ Current Row. Update the database to reflect modifications made to the table in the current row only.
Table space	Leave blank.
16K table space	Leave blank.
32K table space	Leave blank.
Index space	Leave blank.
Table groupings file	Optional. This file is provided by the DBA and is specific to your database.
Privileged user id	Enter your database user ID, for example SADMIN. The table owner is read from <code>tools.cfg</code> .
Privileged user password	Enter your database user password, for example SADMIN. When the database initialization for a mobile client is performed, the table owner changes from SIEBEL to the mobile user's password. In this case, use the mobile user's password in the password field.
ODBC data source	Verify that the ODBC connection specified in the ODBC Data Source text box is correct for your environment. You cannot apply schema changes to any database other than the one you are currently connected to (for example, by specifying the ODBC name of a different database).

NOTE: If you receive an error message and cannot apply changes to the server database, you must use the Database Server Configuration Utility. See [“Applying Schema Changes Using the Configuration Utility” on page 283](#).

6 Restart the Siebel Server.

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

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

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

Applying Schema Changes Using the Configuration Utility

Complete the steps in this section only if you are unable to apply schema changes to the server database as discussed in the previous section.

NOTE: Stop the Siebel Server before running the Database Server Configuration Utility. Otherwise, the task can fail due to a locking issue.

The Configuration Utility performs the following sequence of steps:

- Exports logical schema definition from the specified repository to the .ddl file.
- Synchronizes the physical schema of the application database with this logical schema definition.
- Propagates new repository schema changes to mobile users (if Siebel Anywhere is being used).

To migrate the schema under Windows

- 1** If you are not using Siebel Anywhere, stop all processes running on all Siebel Application Servers, including all Siebel Remote Processes, once all mobile user transactions have been merged and routed.

If you are using Siebel Anywhere, stop all Siebel Application Server processes except the Transaction Preprocessor and Transaction Routers.

- 2** Launch the Database Server Configuration Utility by choosing Start > Programs > Siebel Enterprise Server *version_number* > Configure DB Server.

The Gateway Server Address screen appears.

- 3** Specify your Gateway Server Address and Enterprise Server Name and click Next.

The Installation and Configuration Parameters: Siebel Server Directory dialog box appears.

- 4** In the Siebel Server Directory dialog box, either accept the default value or choose the Browse button to select a directory, and then click Next.

The Siebel Database Server Directory dialog box appears.

- 5** Either accept the default value or choose the Browse button to select a directory, and then click Next.

The RDBMS Platform dialog box appears.

- 6** Choose your db platform and click Next.

NOTE: The default is DB2 UDB.

The Siebel Database Operation dialog box appears.

- 7** Choose Run Database Utilities and click Next.

The Database Utility Selection dialog box appears.

- 8** Choose Synchronize Schema Definition and click Next.

9 The next dialog boxes are described as follows:

- Database Encoding: enter whether or not the database is Unicode or Non-Unicode.
- ODBC Datasource Name dialog box
- Database User Name dialog box: enter user name and password
- Database Table Owner dialog box: enter table owner and password
- Index Table Space Name dialog box:
 - Index 4K tablespace (DB2)
 - 16K/32K tableSpace (DB2)
 - Index tablespace (Oracle)

10 Click Next on these dialog boxes.

The Repository Name dialog box appears.

11 Enter the repository name you wish to synchronize.

NOTE: The default is Siebel Repository.

12 Click Next.

The Configuration Parameter Review dialog box appears. It lists all the parameters you have chosen.

NOTE: Passwords are in asterisks.

13 Click Finish.

The runnow dialog box asks you to verify your choices.

- 14** You can either run the `ddlsync` process now or later. If you choose OK, the Upgrade Wizard is launched. If you choose Cancel, you can run the process later by typing `siebug.exe/m master_ddlsync.ucf` from the `siebsrvr/bin` directory.

If you choose OK, the following Siebel Upgrade Wizard dialog box appears.

- 15** If you choose OK in the Siebel Upgrade Wizard dialog box, the `ddlsync` process starts. It displays your progress by placing a check after each step is completed. If there are any errors during the process, the Upgrade Wizard stops. You can go to the `siebsrvr/log` and look at the log files there. Fix the problem and then relaunch the Upgrade Wizard through `siebug.exe/m master_ddlsync.ucf`.

NOTE: You must verify that there is no `driver_ddlsync` directory under `siebsrvr/upgrade` before you launch the Database Server Configuration Utility. This directory keeps a record of where you are in the `ddlsync` process for restarting. If you have already run `ddlsync`, you need to delete this directory before running `ddlsync` again, or you will receive an error message.

To migrate the schema under UNIX

- 1** Source environment variables from `$SIEBEL_ROOT`.

`SIEBEL_ROOT` should be the path of your Siebel installation directory.

`LANGUAGE` should be set to the language in which the Configuration Wizard prompts appear; for example, `enu` for U.S. English.

If either of these values is incorrect or empty, reset them using one of the following commands, as appropriate to the shell you use:

```
setenv LANGUAGE New Value
```

or

```
export LANGUAGE | SIEBEL_ROOT=New Value
```

- 2** Depending on your shell, enter:

Korn shell

```
export SIEBEL_LOG_EVENTS trace3
```

C shell

```
setenv SIEBEL_LOG_EVENTS trace3
```

NOTE: Setting trace to 3, creates an appropriate level of detail in the log file for this activity.

- 3** Navigate to `$SIEBEL_ROOT /bin` and enter:

```
dbsrvr_config.ksh
```

This launches the Database Server Configuration Wizard.

- 4** Review the values of the following environment variables and confirm whether or not the settings are correct by entering either `Y` or `N`.

NOTE: If either the `SIEBEL_ROOT` or `LANGUAGE` value is not set or is incorrect, you must correct them before proceeding.

- 5** In the Siebel Server Directory dialog box, either accept the default value or choose the Browse button to select a directory, and then click Next.

The Siebel Database Server Directory dialog box appears.

- 6** Either accept the default value or choose the Browse button to select a directory, and then click Next.

The Siebel Database Operation screen appears.

- 7** Choose Run Database Utilities (5).

The Database Utility Selection screen appears.

- 8** Choose Synchronize Schema Definition.

The RDBMS Platform screen appears.

- 9** Choose your db platform.

NOTE: The default is DB2 UDB.

- 10** The next screens are described as follows:

- Database Encoding: enter whether or not the database is Unicode or Non-Unicode.
- ODBC Datasource Name
- Database User Name: enter user name and password
- Database Table Owner: enter table owner and password
- Index Table Space Name:
 - Index 4K tablespace (DB2)
 - 16K/32K tablespace (DB2)
 - Index tablespace (Oracle)

- 11** Press Enter on these screens (leave the parameters blank).

The Repository Name screen appears.

- 12** Enter the repository name you wish to synchronize, or press Enter to use the default (Siebel Repository).

The Configuration Parameter Review screen appears. It lists all the parameters you have chosen.

NOTE: Passwords are in asterisks.

- 13** You can either run the ddlsync process now or later. If you enter Y, the Upgrade Wizard is launched. If you enter N, you can run the process later by typing `siebsrvr.exe/m master_ddlsync.ucf` from the `siebsrvr/bin` directory.

- 14 If you enter Y, the ddlsync process starts. It displays your progress by placing a check after each step is completed. If there are any errors during the process, the Upgrade Wizard stops. You can go to the siebsrvr/log and look at the log files there. Fix the problem and then relaunch the Upgrade Wizard through siebug.exe/m master_ddlsync.ucf.

NOTE: You must verify that there is no driver_ddlsync directory under siebsrvr/upgrade before you launch the Database Server Configuration Utility. This directory keeps a record of where you are in the ddlsync process for restarting. If you have already run ddlsync, you need to delete this directory before running ddlsync again, or you will receive an error message.

Applying Server Database Changes to Other Local Databases

At this point, your extensions have been applied to your server database, and exist on the physical database. Next, you must propagate the schema changes to mobile users.

To propagate schema changes to mobile users

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

Using Siebel Anywhere. Perform the following steps:

- a Create an Upgrade Kit on your Server database that includes the Siebel Database Schema as the upgrade kit component. Refer to *Siebel Anywhere Guide* for information on creating upgrade kits.
- b Click Activate on the Upgrade Kits View to make the upgrade kit available.

Without Siebel Anywhere. Perform the following steps:

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

- d** Click Activate in the Table List view. Executing this process will increase the database schema version number, and therefore require a schema upgrade for mobile users.
- 3** Run `gennewdb` to regenerate the template local database.

The Server Administration screen in the Siebel client has a view where this job can be run. You can also use the `svrvmgr.exe` utility through a DOS prompt. See *Siebel Remote and Replication Manager Administration Guide*.

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

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

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

Populating Extension Tables and Columns

You can load initial data into your extension tables and columns by using a view where the new fields are displayed in an applet. If you have a large amount of data to load, or if the user interface does not permit data entry, you may be able to use EIM or client-side import to load data into the extension table or column.

Making Extension Tables Available for Population by EIM

To understand how to modify your data schema to use EIM, see [Chapter 5, “Data Objects Layer.”](#) Work with your database administrator to populate the interface table with the extension column data and matching user keys. Run EIM to import the data from the interface table into the target tables.

To make an extension table available for population by EIM

- 1** Lock the EIM Interface Table project.

- 2** Navigate to the EIM Interface Table object, create a new record, and then complete its properties, for example Name and Target Table Name.
- 3** Click Activate in the EIM Tables list.

The temporary columns required to populate the extension table are created.
- 4** Navigate to the Table object, and then select the EIM interface table created in [Step 2](#).
- 5** Click Apply in the Tables list, complete the fields in the Apply Schema dialog as shown in [“To update your local environment” on page 275](#).

The generated temporary columns are applied to the database.
- 6** Click Activate in the Tables list.

The extension table is available to be populated by EIM.

Configuring Client-Side Import

You can use client-side import to populate fields with data. Client-side import uses the applet-level menu import functionality in the Siebel Web Client. It is configured using the Import Object object in Siebel Tools, which sets business component fields to be populated. Client-side import is supported for parent business components only. Applets configured for client-side import must allow records to be inserted; that is, the *No Insert* property of the applet must be set to *False*.

For an example of how client-side import is configured, see the Contact business component in the standard Siebel repository. The contact business component is configured as an Import Object with fields defined as Import Field child objects.

To enable client-side import for a business component

- 1** Lock the project to which the business component belongs.
- 2** In the Object Explorer, select Import Object.
- 3** Add a new record with the business component and its project as properties.
- 4** With the new record selected, expand Import Object, and then select the Import Field child object.

- 5 Add new records for each business component field you wish to be populated.

NOTE: You can also add import fields to already existing import objects, such as Contact.

- 6 Compile the .srf, selecting the locked project.

The new fields will be displayed in the Select a Siebel Field dialog and can be mapped to fields in the External Data Source Field dialog when importing data.

Advanced Database Extensibility

The Advanced Database Extensibility option allows you to create new tables and enable them for docking and EIM processes. You can add the following types of tables to the Siebel Data Model:

- **Stand-Alone Table:** A new table that is not related to any existing table in the Siebel Data Model.
- **1:1 Extensions to Existing Tables:** A new table that bears a 1:1 relationship to an existing Base Table in the Siebel Data Model.
- **M:1 Extensions to Existing Tables:** A new table that bears a Many: 1 relationship to an existing Base Table in the Siebel Data Model. This table closely resembles a _XM table in the current Siebel Data Model.
- **Intersection Tables between Existing Tables:** An intersection table (Many: Many) between two existing tables.

The Table Wizard allows you to create the types of tables described above.

NOTE: Changes made using the Advanced Database Extensibility module are upgradable. Before undertaking database extensions and modifications using Advance Database Extensibility module, make sure that all other extensibility options have been considered.

The EIM Mapping wizard allows you to create or associate the new table to the appropriate interface tables for using EIM and to import data into custom tables using EIM. The Docking Wizard allows you to associate the new table with an existing or new Dock object in order to support the synchronization of the contents of custom tables to remote users.

Creating New Tables Using the Table Wizard

The Table Wizard allows you to create new stand-alone tables, extension tables, and intersection tables. It provides picklists with appropriate choices for each type of table and makes sure that the naming conventions are observed. You can only create tables of types: Data (Public), Data (Intersection), and Data (Extension). You must explicitly grant permissions on any table that you create.

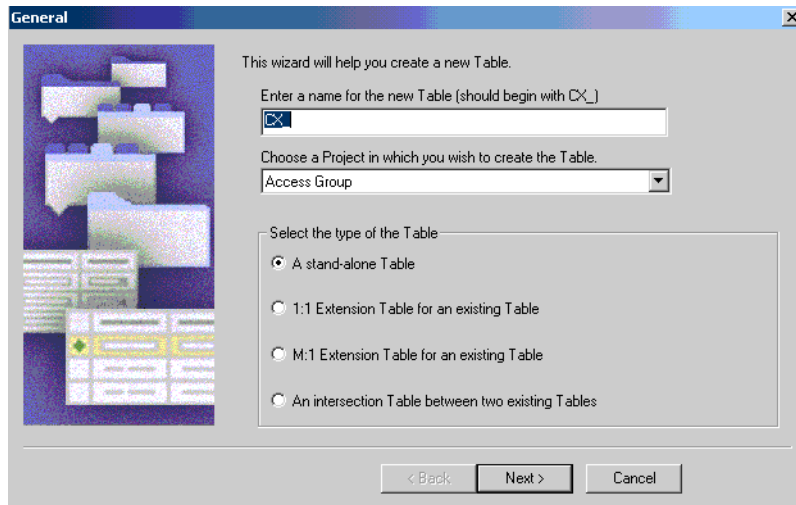
To use the Table Wizard

- 1** Choose File > New Objects.

The New Objects dialog box appears.

- 2 Select the Table Wizard icon.

The General dialog box appears.



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

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

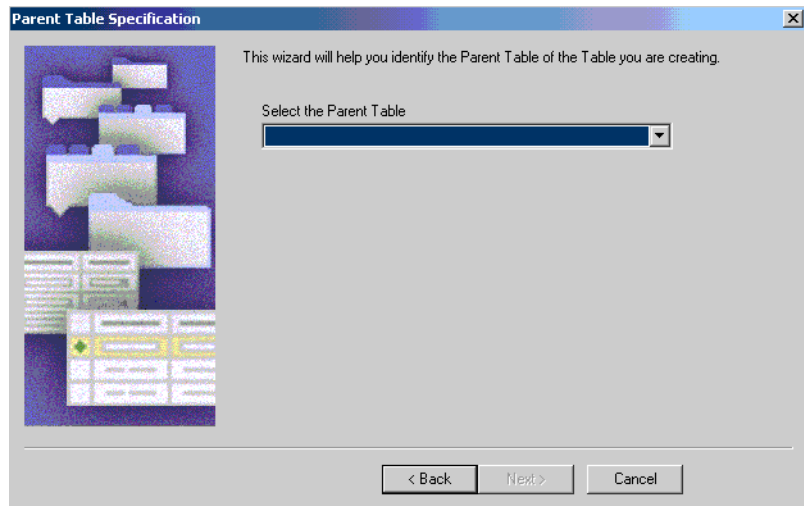
- 4 In the “Choose a Project in which you wish to create the Table” field, choose a project.

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

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

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

- a If you choose Stand-alone Tables, the Parent Specification Table dialog box is not displayed and you are taken to the Finish dialog box, stating that the new table can now be created.
- b For 1:1 Extension Tables and M:1 Tables, the Parent Specification Table dialog box allows you to select the parent table. See the following figure:



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

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

NOTE: Users can add new columns to S_CONTACT, S_ORG_EXT, S_POSTN, or S_USER, for example (and this is generally preferable to adding new columns to S_PARTY); they cannot create new 1:1 relationships to them.

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

Parent Table Specification

This wizard will help you identify the Parent Tables and Foreign Keys of the Table you are creating.

Select the first Parent Table.

Enter a Foreign Key Column name for the first Parent Table.

Select the second Parent Table.

Enter a Foreign Key Column name for the second Parent Table.

< Back Next > Cancel

- The picklist for the “Select the first Parent Table” field is restricted to all tables of type Data (Public).
- The picklist for the “Select the second Parent Table” field is restricted to tables of type Data (Public) with the following added restrictions, based on the choice of the first parent table.
- The names of the Foreign Key columns (“Enter a Foreign Key Column name for the first Parent Table” field and “Enter a Foreign Key Column name for the second Parent Table” field) are verified to make sure that they are unique (that is, do not conflict with each other or the system column names).

- 7 Click Next on the Parent Table Specification dialog box.

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

- 8 Click Finish to generate the table.

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

Table Wizard Actions

The following columns are generated by the Table Wizard.

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

- Type = User Key
 - User Primary Key = TRUE
- M1 index on TYPE (1) and NAME (2)
 - Unique/Cluster = FALSE
 - Type = System
- P1 index on ROWID
- For intersection tables, the Table Wizard sets the type of the table to Data(Intersection) and creates the following:
 - TYPE column for added user functionality
 - Two Foreign Key columns with names specified in the Table Wizard
 - User Key Sequence = 1 and 2
 - Foreign Key Table = < Parent Table >
 - U1 index on the two Foreign Keys (1, 2), TYPE (3), and CONFLICT_ID (4)
 - Unique/Cluster = TRUE
 - Type = User Key
 - User Primary Key = TRUE
 - F1 index on the Foreign Key to the second parent table

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

EIM Table Mapping Wizard

The EIM Table Mapping Wizard is accessed by right-clicking on a table entry in the Table list applet. The EIM menu item is only enabled with the non-standalone customer table. You can select an interface table and specify a prefix for the interface table columns that are generated by the wizard and that create a complete hierarchy of EIM Table Mapping objects needed to import and export data to and from the selected table. See [Figure 74 on page 300](#).

The EIM Table Mapping Wizard does not map foreign key columns if they point to a parent table column that does not have user key attributes. Because you cannot add user key attributes, this means that the EIM Table Mapping Wizard does not work with either standalone tables or foreign keys to custom tables.

In order to invoke the EIM Table Mapping Wizard for those Siebel base tables that do not have the foreign key as part of the user key, you need to create a temporary column with the following properties:

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

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

NOTE: The EIM Table Mapping Wizard is only available to tables of type Data (Public), Data (Intersection), Data (Extension), and Data (Extension-Siebel).

The EIM Table Mapping Wizard is only available to tables of type Data (Public), Data (Intersection), Data (Extension), Data (Extension-Siebel).

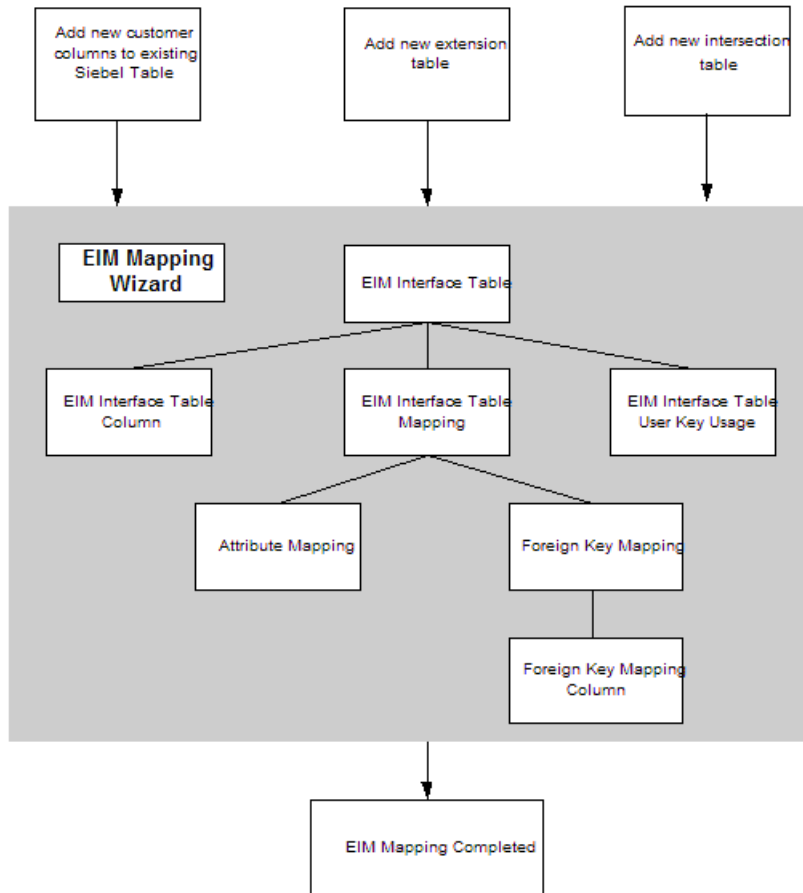


Figure 74. EIM Mapping

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

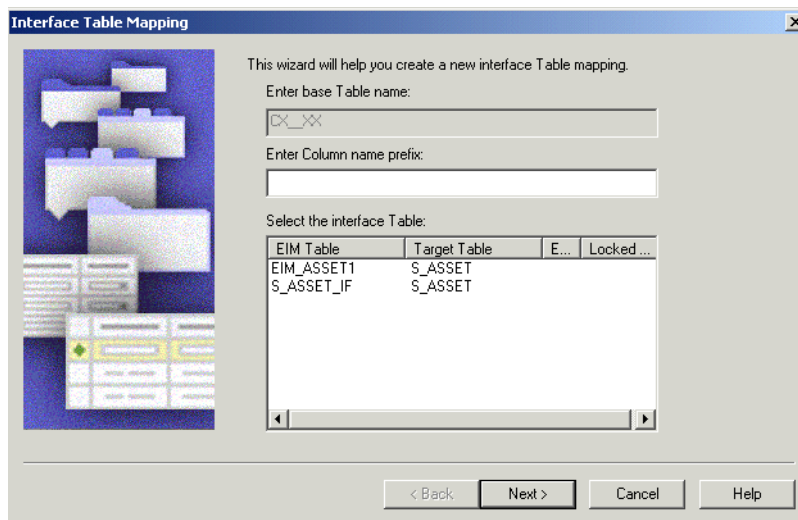
- 1** Lock the project.
- 2** Select Table object type in the Object Explorer.

3 Select an entry in the Object List Editor.

Choose from the list a base table that will be mapped to an EIM Table. It will be the primary table into which data from the existing Interface table will be imported.

4 Right-click and select EIM Mapping Table from the menu.

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



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

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

- 6 In the *Select an interface table* field, select from the picklist.

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

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

- 7 Click Next on the Interface Table Mapping dialog box.

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

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

Based on this information, the wizard creates new EIM table mapping objects and adds several child objects to an existing EIM interface table object:

- EIM Interface Table Column
- EIM Table Mapping
 - Attribute Mapping
 - EIM Explicit Primary Mapping
 - Foreign Key Mapping
 - Foreign Key Mapping Column

EIM Object Specifications

The specifications for EIM objects are as follows:

■ **EIM Interface Table** ([Table 14](#))

Table 14. EIM Interface Table Object Specifications

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

■ **EIM Interface Table Columns** ([Table 15](#))

Table 15. System Columns on the EIM Interface Table

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

■ Generic EIM Interface Table Columns for EIM processing

For each EIM Table Interface the following three columns are created to facilitate processing. Customers cannot change their values, which are shown in [Table 16](#).

Table 16. Generic EIM Interface Table Columns for EIM Processing

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

■ **EIM Interface Table Columns for processing a mapping to a particular table**

There are four of these columns for each EIM Table Mapping object, with the following properties (Table 17):

Table 17. EIM Interface Table Columns for Processing a Particular Mapping

Column	Value
Name	Derived from the name of the target table: “T_” + [EIM Table Mapping Name without the “CX_”] + “_” + [Process-specific suffix]
Physical Type	Depends on the process to which the column is related
Length	Depends on the process to which the column is related
Type	Depends on the process to which the column is related
User Name	Name of the EIM Table Mapping object for which the column is being created
EIM Processing Column	TRUE

For example, if the target table selected by the user is CX_SEC_LEV, an EIM Table Mapping object is created. The following four column objects are generated, with the corresponding default properties (Table 18):

Table 18. EIM Interface Table Columns for Processing a Mapping to CX_SEC_LEV

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

- **EIM Interface Table Columns for foreign key processing**

A column is created for each foreign key on the relevant EIM Table Mapping object (that is, the target table). These columns have the following properties (Table 19):

Table 19. EIM Interface Table Columns for Foreign Key Processing

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

For example, if CX_SEC_LEV contains foreign key columns called OPTY_ID and ACCNT_ID to the S_OPTY and S_ORG_EXT tables, respectively, the following EIM Table Columns are generated (Table 20):

Table 20. EIM Interface Table Columns for Processing Foreign Keys in CX_SEC_LEV

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

■ **EIM Interface Table Columns for foreign keys**

A separate foreign key column will be created for each U1 user key column on the foreign key tables. The columns have the following properties (Table 21):

Table 21. EIM Interface Table Columns for Foreign Keys

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

Continuing with the CX_SEC_LEV example (Table 22):

Table 22. EIM Interface Table Columns for CX_SEC_LEV Foreign Keys

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

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

- **EIM Interface Table Column for each attribute on the target table**

Attribute columns on the target table are those of type Data (Public) that have a null Foreign Key Table property. These EIM interface table columns will have the following properties (Table 23):

Table 23. EIM Interface Table Columns for Each Attribute on the Target Table

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

For example, if you enter a prefix of SECL and have the following attribute columns in CX_SEC_LEV: NAME (Varchar 100), DESC_TEXT (Varchar 250), and AUTO_UPDATE (Char 1), the following EIM interface table columns are generated (Table 24):

Table 24. EIM Interface Table Columns for Each Attribute on CX_SEC_LEV

Name	Physical Type	Length	Type	User Name
SECL_NAME	Varchar	100	Data (Public)	Security Level Name
SECL_DESC_TEXT	Varchar	250	Data (Public)	Security Level Description
SECL_AUTO_UPDATE	Char	1	Data (Public)	Auto Update Flag

■ **EIM Table Mapping objects based on the target table**

The name and destination columns will be set to the name of the target table. The processing column properties will correspond to those that have been automatically generated. For example (Table 25):

Table 25. EIM Table Mapping Objects Based on the Target Table

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

■ **Attribute Mapping objects for each EIM interface column generated**

These should have the following property values (Table 26):

Table 26. Attribute Mapping Objects for Each EIM Interface Column

Object	Value
Name	Attribute column on target table
Interface Table Data Column	Name of corresponding EIM interface table column generated (Table 24 on page 308)
Base Table Attribute Column	Name of attribute column on target table

- **Foreign Key Mapping for each foreign key column on the Target Table**

A separate Foreign Key Mapping object is created for each foreign key mapping column on the target table. The following properties are set for each (Table 27):

Table 27. Foreign Key Mapping for Each Foreign Key Column on a Target Table

Object	Value
Name	Name of the user key column
Foreign Key Column	Name of the user key column
User Key	Name of the U1 user key of the foreign key table
EIM Foreign Key Proc Column	Corresponding EIM interface table column for foreign key processing: “T_” + [Target table name without “CX_” prefix] + “_” + [user key column name]

Continuing with the CX_SEC_LEV example (Table 28):

Table 28. Foreign Key Mapping for Each Foreign Key Column on CX_SEC_LEV

Name	Foreign Key Column	User Key	EIM Foreign Key Proc Column
OPTY_ID	OPTY_ID	S_OPTY_U1	T_SEC_LEV_OPTY_ID
ACCNT_ID	ACCNT_ID	S_ORG_EXT_U1	T_SEC_LEV_ACCNT_ID

■ **Foreign Key Mapping Columns for each Foreign Key Mapping object**

A separate Foreign Key Mapping Column object is created for each user key column in the user key specified for the parent Foreign Key Mapping object (Table 27 on page 310). The following properties are set for each foreign key mapping column (Table 29):

Table 29. Foreign Key Mapping Columns for Each Foreign Key Mapping Object

Column	Value
Name	EIM interface table column to which the user key column on the target table should be mapped. This EIM interface table column is generated according to the specifications in Table 21 on page 307.
Interface Data Column	EIM interface table column to which the user key column on the target table should be mapped. This EIM interface table column is generated according to the specifications in Table 21 on page 307.
User Key Attribute	Name of the corresponding user key column that belongs to the user key specified in Table 27 on page 310.

Continuing with the CX_SEC_LEV example (Table 30):

Table 30. Foreign Key Mapping Columns for CX_SEC_LEV

Name	Interface Data Column	User Key Attribute
OPTY_BU_ID	OPTY_BU_ID	BU_ID
OPTY_NAME	OPTY_NAME	NAME
OPTY_PR_DEPT_OU_ID	OPTY_PR_DEPT_OU_ID	PR_DEPT_OU_ID
ORG_BU_ID	ORG_BU_ID	BU_ID
ORG_NAME	ORG_NAME	NAME
ORG_LOC	ORG_LOC	LOC

Dock Objects

Siebel applications can selectively replicate server data into the local database, and thus support mobile computing by allowing field personnel to share information across the enterprise, including both mobile and connected users. This is accomplished using Siebel Remote.

Dock objects are the foundation of Siebel Remote. Dock objects are collections of related tables. Each Dock Object object in Siebel Tools has one primary table and several other related tables as Dock Object Table child objects.

Different data is replicated to different local databases, depending on each local database owner's employee identity, position, organization, and visibility to data from different dock objects, and on the relationship between the dock objects. These rules are known collectively as routing rules or Dock Object Visibility Rule child objects.

When the data is updated on the server, the local database is synchronized when the mobile user connects to the Siebel Server and performs the synchronization, but only the data that should be replicated into the local database is synchronized. During the synchronization, any updates in the local database are also uploaded to the server.

For more information on routing rules and synchronization, see *Siebel Remote and Replication Manager Administration Guide*.

Dock Object Types

There are three types of dock objects:

- **Private.** Private dock objects are used exclusively for routing of non-configurable data. This setting makes sure that the rows in these dock objects are never routed to any mobile clients. All records from tables that are part of a private dock object are uploaded to the server during synchronization. None of these records are downloaded to remote users.

- **Enterprise.** Enterprise dock objects involves a distribution of records without restriction. All records from tables that are part of an Enterprise dock object are uploaded to the server during synchronization. Most of these tables should only be updated by an administrator and are typically downloaded but not uploaded by mobile users. To minimize synchronization time, Enterprise dock objects should only be used with tables that contain small volumes of data or are semi-static in nature (that is, the contents change infrequently).
- **Limited.** Limited dock objects contain numerous individual rules for determining the records that should be downloaded to particular users; these users should only get those records with which they have some direct or indirect involvement.

There are nine types of visibility rules used for Limited dock objects:

- **Employee.** Evaluates record according to whether it has a foreign key to the mobile user's Employee record.
- **Employee Manager.** Evaluates record according to whether it has a foreign key to the Employee record of someone who directly reports to the mobile user.
- **Position.** Evaluates record according to whether it has a foreign key to the mobile user's primary Position record.
- **Position Manager.** Evaluates record according to whether it has a foreign key to the Position record of someone who directly reports to the mobile user.
- **Organization.** Evaluates record according to whether it is associated with the same business unit as the mobile user.
- **Check Dock Object.** Evaluates record according to whether it is related to another record that the user receives.
- **Calendar.** Applies only to calendar appointment records. Evaluates record according to whether the mobile user has access to the calendar of the record's owner.
- **Category.** Evaluates record according to whether it is in a category visible to the user.
- **SQL Rule.** Used to handle special exceptions through custom SQL.

For more information, see [“Dock Object Visibility Rules” on page 316](#).

Dock Object Tables

The Dock Object Table object type is a child object type of Dock Object, and is used to specify the tables whose records are actually transferred in conjunction with the Dock Object. The Opportunity dock object and its child dock object tables are shown in [Figure 75](#).

W	Name	Changed	Project	Inactive	Active
>	Opportunity		Dock Opportunity		✓

W	Table Name	Visibility Strength	Source Column Name	Target Table Name	Target Column Name
>	S_NOTE_OPTY	100	SRC_ROW_ID	S_OPTY	ROW_ID
	S_OPTORG_CMPT_I	100	PAR_ROW_ID	S_OPTY_ORG_CMPT	ROW_ID
	S_OPTY	25			
	S_OPTY_ASSET	50	OPTY_ID	S_OPTY	ROW_ID
	S_OPTY_ATT	100	PAR_ROW_ID	S_OPTY	ROW_ID
	S_OPTY_BU	50	OPTY_ID	S_OPTY	ROW_ID
	S_OPTY_CHRCTR	100	OPTY_ID	S_OPTY	ROW_ID
	S_OPTY_CNPRDINT	50	OPTY_ID	S_OPTY	ROW_ID
	S_OPTY_COMM	100	OPTY_ID	S_OPTY	ROW_ID
	S_OPTY_CON	50	OPTY_ID	S_OPTY	ROW_ID
	S_OPTY_INDUST	50	OPTY_ID	S_OPTY	ROW_ID
	S_OPTY_ISS	100	OPTY_ID	S_OPTY	ROW_ID
	S_OPTY_MKT_SEG	50	OPTY_ID	S_OPTY	ROW_ID
	S_OPTY_ORG	50	OPTY_ID	S_OPTY	ROW_ID
	S_OPTY_ORG_CMPT	100	OPTY_ID	S_OPTY	ROW_ID
	S_OPTY_POSTN	50	OPTY_ID	S_OPTY	ROW_ID
	S_OPTY_PROD	50	OPTY_ID	S_OPTY	ROW_ID
	S_OPTY_PROJ	100	OPTY_ID	S_OPTY	ROW_ID
	S_OPTY_REL	100	PAR_OPTY_ID	S_OPTY	ROW_ID

Figure 75. Dock Object and Dock Object Tables

All of the tables identified in dock object tables for a given dock object are related, through foreign keys in the data model, to one *driving table* (also represented by a Dock Object Table object definition). The driving table is identified in the Primary Table property in the Dock Object object type.

For example, the Opportunity dock object shown in [Figure 75 on page 314](#) is based on the primary table S_OPTY, but it also includes other dock object tables such as S_NOTE_OPTY (notes for the opportunity) and S_OPTY_REL (relationships between opportunities). Also included are the extension tables for S_OPTY.

A dock object is therefore a set of logical records (opportunities in this case), where each such logical record is itself a collection of one or more physical database records spread across multiple tables.

Dock Object Visibility Rules

To determine which records in a dock object to download to each mobile user, the Siebel application evaluates the dock object visibility rules for that dock object. Dock Object Visibility Rule is a child object type of Dock Object, as illustrated for Opportunity in [Figure 76](#).

The screenshot displays two windows from Siebel Tools. The top window, titled "Dock Objects", contains a table with the following data:

W	Name	Changed	Project	Inactive	Active	Visibility Level	User Name
>	Opportunity		Dock Opportunity		✓	Limited	Opportunity

The bottom window, titled "Dock Object Visibility Rules", contains a table with the following data:

W	Sequence	Visibility Strength	Description	Type	Check	Comments
>	1	100		Position		You are on the sales team of the Opportunity
	2	100		Position Manager		You are the manager of the primary sales rep on the Opp
	3	50		Check Dock Object	100	Related Opportunity for an Opportunity you have full vis
	4	50		Check Dock Object	100	Opportunity for an Opportunity Forecast you have full v
	5	50		Check Dock Object	100	Opportunity for an Activity you have full visibility on
	6	50	S_OPTY.PR_DEPT_OU_ID is not a prir	Check Dock Object	100	Opportunity for an Account you have full visibility on
	7	50		Check Dock Object	100	Opportunity for an Indirect Account you have full visibi
	8	50		Check Dock Object	100	Opportunity for a Contact you have full visibility on
	9	50		Check Dock Object	100	Opportunity for an Agreement you have full visibility on
	10	50		Check Dock Object	100	Opportunity for a Quote you have full visibility on
	12	50	Deactivated on 5/22/01.	Check Dock Object	100	Opportunity for a ContactProduct you have full visibility
	13	100	Inactivated due to slow performance.	Position Manager		You are the manager of ANY sales rep on the Opportunit
	14	50		Check Dock Object	100	Opportunity for an Opportunity Product Forecast you h
	15	50		Check Dock Object	100	Opportunity for an Opportunity Position Forecast you hi
	16	50		Check Dock Object	100	Opportunity for an Opportunity Product Position Foreca
	17	50	Deactivated on 5/22/01.	Check Dock Object	100	Opportunity for a Communication that you have full visi
	18	50		Check Dock Object	100	Source of your f.v. project
	19	50		Check Dock Object	100	Pursued by your f.v. project
	20	100	Inactivated because it's redundant.	Employee		You are the primary sales rep on the Opportunity (emplo

Figure 76. Dock Object and Dock Object Visibility Rules

Each visibility rule has a Comment property that explains specifically what the rule checks. For example, the dock object visibility rules on the Opportunity dock object include the following: “You are on the sales team of the Opportunity,” “You are the manager of the primary sales rep on the Opportunity’s sales team,” and so on. In addition, each dock object visibility rule has a Visibility Strength property and a Sequence property.

Siebel applications determine which database records to propagate to each mobile user (for dock objects that have limited visibility) by evaluating the *visibility strength* of the user for a dock object, and comparing this with the visibility strengths of the tables it contains.

The user's visibility strength for a dock object is determined from the child dock object visibility rules. For each dock object record, and for each mobile user, the Siebel application sequentially evaluates the rules in order of descending visibility strength and ascending sequence until one of them "passes" (that is, evaluates to TRUE). As soon as one of the rules passes, the Siebel application stops this evaluation process, and gives the current dock object record to the current mobile user.

When a dock object visibility rule passes, the mobile user gets the parent dock object record with a visibility strength value obtained from the corresponding property in the dock object visibility rule that caused him or her to get the record. If none of the dock object visibility rules passes for a given dock object record and a given mobile user, then that user will not receive that particular record.

For example, consider two different dock object visibility rules in the Opportunity dock object:

- The first visibility rule ("You are on the sales team of the Opportunity") has a visibility strength of 100.
- The sixth rule ("Opportunity for an Account you have full visibility on") has a visibility strength of 50.

If users are on the sales team for a particular opportunity, they get that opportunity record with a visibility strength of 100. However, if they are not on the sales team for that opportunity—and if the next four visibility rules also fail—they still get the opportunity record with a visibility strength of 50 if they have full visibility to the account for that opportunity.

Visibility strength values are integers between 0 and 100. A visibility strength of 100 denotes full visibility, while a visibility strength of 0 denotes no visibility. Any value between 1 and 100 (typically 25 or 50) implies partial visibility.

NOTE: The integer range for a visibility strength value is actually 0–254, but a value of 100 is, by convention, considered to mean full visibility. If your configuration does not require the use of values higher than 100, use values in the 0–100 range rather than 0–254.

The user's visibility strength (obtained from the successful dock object visibility rule) is compared with each dock object table's visibility strength, as specified in its Visibility Strength property. For users to receive the records from a particular dock object table, their visibility strength must be greater than or equal to the visibility strength specified for that table.

For example, suppose that a particular mobile user receives a particular logical record from the Opportunity dock object with a visibility strength of 50. The Siebel application will then propagate to the user's local database all physical records that are related to the given opportunity on any of the dock object tables that have a visibility strength less than or equal to 50 in the Opportunity dock object.

For more information on routing (visibility) rules and their implementation, see the chapter about Siebel Remote administration in *Siebel Remote and Replication Manager Administration Guide*.

Finding the Dock Object for a Business Component

Dock objects are provided for standard Siebel applications. Review the standard dock objects and associated visibility rules thoroughly to see if they satisfy a desired visibility change.

Some important business components and their associated dock objects are shown in [Table 31](#).

Table 31. Important Business Components and Their Dock Objects

Business Component	Dock Object	Primary Table	Visibility Level
Action	Activity	S_EVT_ACT	Limited
Account	Party	S_PARTY	Limited
Asset Mgmt - Asset	Asset	S_ASSET	Limited
Contact	Party	S_PARTY	Limited
Employee	Party	S_PARTY	Limited
Opportunity	Opportunity	S_OPTY	Limited
Position	Party	S_PARTY	Limited
Internal Product	Product	S_PROD_INT	Limited
Service Request	ServiceRequest	S_SRV_REQ	Limited

NOTE: Although the Employee and Position records are represented by the Party dock object, which is of Limited visibility, the employee and position records themselves have Enterprise visibility, based on SQL Rule type rules within the Party dock object.

To determine the dock object to which a business component belongs

- 1 In the Object Explorer, select the Business Component object, and then query for the desired business component.
- 2 Note the Table property of the business component.

For example, the base table of Opportunity is S_OPTY.

- 3 In the Object Explorer, select the Flat tab, select the Dock Object object, and then query in the Dock Object Table field for the table.

If the table belongs to a dock object, that dock object will be listed. For example, the dock object to which the Opportunity business component belongs is Opportunity.

Docking Wizard

The Docking Wizard is used to extend Siebel Remote functionality to support custom database schema changes. You can use the Docking Wizard to do the following:

- Create new dock objects for custom extension tables that are not already in a dock object.
- Create new dock object tables for custom dock objects.
- Create new dock object visibility rules for custom and existing dock objects.

NOTE: This is not done directly by the user. The appropriate visibility rules will be added to the dock object depending on the visibility type of the dock object and the structure of the tables involved.

New dock object visibility rules can be added as a result of one of two actions:

- A table is added to a custom dock object as a dock object table using the Docking Wizard.
- The Docking Wizard is invoked from a custom extension column that acts as a foreign key to another table.

The Docking Wizard automatically creates or updates Dock Object, Dock Object Table, and Dock Object Visibility Rule objects for custom tables. You can create Public, Private, and Limited dock objects through the Docking Wizard.

The Docking Wizard creates Limited dock object visibility rules of the following types:

- **Employee.** Employee rules replicate data depending on the mobile user's employee identity. To find all candidate rules, find all columns that are foreign keys to S_USER table, except CREATED_BY and LAST_UPD_BY.
- **Employee Manager.** Employee Manager rules replicate data based on which employees report to the mobile user. The algorithm for finding all candidate rules is the same as for Employee rules.
- **Position.** Position rules replicate data based on the position the mobile user holds. To find all candidate position rules, the algorithm finds all columns that are foreign keys to the S_POSTN table.
- **Position Manager.** Position manager rules replicate data based on which positions report to the mobile user position. The algorithm to find all candidate rules is the same as for Position rules.
- **Check Dock Object.** Check Dock Object rules replicate data depending on which piece of data from other dock objects is replicated to the mobile local database. The relationship between data in other dock objects and the current dock object determine which records from the current dock object are replicated.

The Docking Wizard can only find the candidate Check Dock Object rules based on the "Foreign Key Table Name" property definitions for columns. For each foreign key, there are two candidate Check Dock Object rules, regardless of where the foreign key column resides:

- Rules that use this dock object as the destination dock object. There are two types of these rules:
 - **Based on foreign keys on the primary table of the current dock object.** The algorithm to find this kind of candidate rule must find in the table of the current dock object all foreign key columns, other than those pointing to S_USER or S_POSTN. For these foreign key columns, the algorithm needs to find the foreign key table to which these foreign key columns refer. The dock object of the foreign key table will become the Check Dock Object object of the newly created Check Dock Object rule in the current dock object.

- **Based on foreign keys on the primary table of other dock objects.** To find this type of candidate rule, the algorithm must find all foreign key columns that refer to the primary table of the current dock object, on any table that is part of a limited dock object. The algorithm will add the appropriate Check Dock Object visibility rules to these limited dock objects, with the current dock object being the Check Dock Object object.
- Rules that use this dock object as the source dock object, that is, Check Dock Object rules. There are two types of these rules:
 - Based on foreign keys on the primary table of the current dock object.
 - Based on foreign keys on the primary table of other dock objects.

The algorithm for these types of rules is similar to the algorithm for rules that use this dock object as the destination dock object. The main difference involves switching the source table or column and target table or column.

The Docking Wizard process flow is shown in [Figure 77](#).

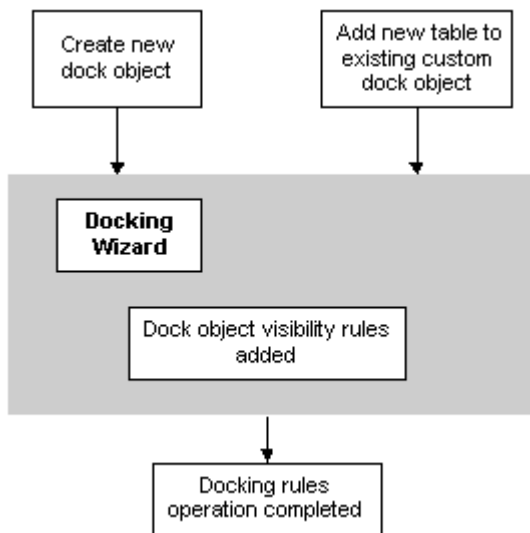


Figure 77. Docking Wizard Flow

Before using the Docking Wizard, you should be aware of the following considerations:

- You can only invoke the Docking Wizard on custom extension tables that are not already placed into any dock objects.
- You cannot invoke the Docking Wizard on standard Siebel out of the box tables. However, you can invoke the Docking Wizard from a custom extension column that has been added to a standard table.
- For a custom table that has a mandatory foreign key to another custom table that is already in a custom dock object, you can either create a new dock object for it or add it to the existing custom dock object. The approach depends on the business requirements and desired outcome.
- The rules created by the Docking Wizard will have a check dock object visibility strength of 100 and a visibility strength of 50. These strengths can not be modified without engaging Expert Services.
- When custom tables are added as dock object tables, they are added with a visibility strength of 50. If this is not appropriate for the business requirements, Expert Services needs to be engaged to modify this strength.

The Docking Wizard can be invoked from two places:

- Table
- Column

To invoke the Docking Wizard from a table

- 1** Select a custom extension table.
- 2** Lock the project.
- 3** Right-click the table record, and then choose Docking Wizard from the pop-up menu.

The Docking Wizard will not be activated if the table already exists in a dock object. If the Docking Wizard completes successfully for the table, the Docking Wizard will not be activated again on right-clicking the mouse.

To invoke the Docking Wizard from a column

- 1 Select a custom extension column.

Custom extension columns have the prefix X_.

- 2 Lock the project.

- 3 Right-click the column record, and then choose Docking Wizard from the pop-up menu.

The Docking Wizard is activated if the column name is prefixed by X_ or the table name is prefixed by CX_ and the table is already in a dock object, whether the table is a Siebel table or a custom table.

The Docking Wizard can be invoked multiple times, regardless of whether it has been run for this column before or not.

The behavior of the Docking Wizard differs depending on where it is invoked:

- From a table:
 - If the custom table is standalone, the only option is to create a new dock object for it. After the dock object creation, appropriate routing rules will also be created.
 - If the custom table has appropriate foreign keys to other custom tables (excluding Siebel tables) already in certain dock objects, there are two options. Appropriate routing rules will be created for either option.
 - Create a new dock object.
 - Add the table to an existing custom dock object.

- From a column:

You do not need to make any choices. The Docking Wizard will add appropriate routing rules:

- For a regular foreign key, two Check Dock Object routing rules will be added: one from the table's dock object to the foreign key table's dock object and the other in the opposite direction.
- For a foreign key to S_POSTN, only a position rule will be added.

Creating a New Dock Object

If you create a new dock object for a standalone customer table, you need to lock the project where you want the new dock object to reside. If you create a new dock object for a non-standalone customer table, you not only need to lock the project where you want the new dock object to reside, but also lock all the projects containing the dock objects in which the parent tables of the customer table reside.

If you invoke the Docking Wizard from a standalone custom table, only the Create a New Dock Object option is activated. The Add the Table to an Existing Dock Object option will be deactivated. When the Docking Wizard creates rules, it creates rules on associated dock objects. If any other project needs a rule added to a dock object in that project, a dialog box appears warning you that other projects need to be locked, if they are not already locked.

NOTE: You must select a custom extension table (that is, beginning with CX_), and then right-click to access the Docking Wizard.

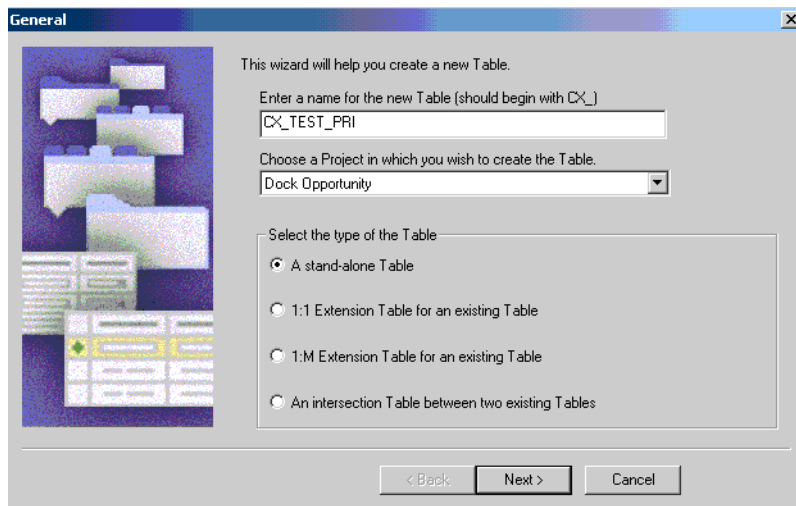
To create a table for the new dock object

- 1** Lock the project that will contain your new table, for example Newtable.
- 2** Select File > New Object.

The New Object dialog box appears.

- 3** Select the Table icon under the General tab.

- 4 In the first General dialog box enter the name of your new table beginning with CX_, select the project, and then select the radio button for the type of table you want.



For example, create a standalone custom extension table called CX_TEST_PRI.

- 5 Click Next.

The Finish dialog box with your entries appears.

- 6 Click Finish to accept the entries.

You are taken to the Tables Object List Editor, where you see your new table displayed.

To create the new dock object

- 1 Lock the project that will contain the dock object, for example Dock Opportunity.
- 2 In the Tables Objects List Editor, select the table for which you want to create a dock object (for example, CX_TEST_PRI created above).

- 3 Right-click, and then select Docking Wizard from the menu options.

NOTE: The Docking Wizard can be launched from entries in the Tables Object List Editor in Table > Column, Table > Index, or Table > User Key object types.

The Add Table to Dock Object dialog box appears.

NOTE: If you invoke the Docking Wizard from a standalone table, only the Create a New Dock Object option is activated. The Add the Table to an Existing Dock Object option is deactivated. When the Docking Wizard creates rules, it creates rules on associated dock objects. If a rule needs to be added to a dock object in a different project that is unlocked, a dialog box appears warning you that other projects should also be locked.

- 4 In the Dock Object field, enter the name of the dock object, for example, DOX PRI.

NOTE: This field must be populated with the DOX prefix.

- 5 In the Project field, all locked projects are listed in the picklist.
- 6 Choose the project for the dock object.
- 7 In the Visibility level section, choose Private, Enterprise, or Limited.

NOTE: If you chose Limited, the employee, employee manager, position, and position manager rules are created on the new dock object, depending on the structure of the table. Dock object rules are created on both the new dock objects and the parent tables' dock objects.

- 8 Click Next.

The Summary page appears.

- 9 If the information displayed is correct, click Finish.

The Docking Wizard creates the new dock object.

Adding a New Dock Table to an Existing Custom Dock Object

Your new table should be a dock object table of an existing dock object. In this situation, the new table is a child of an existing dock object table. Mobile users receive records in the new table if they have access to its parent record in the existing table.

NOTE: You must select a custom extension table (that is, beginning with CX_) and right-click to access the Docking Wizard.

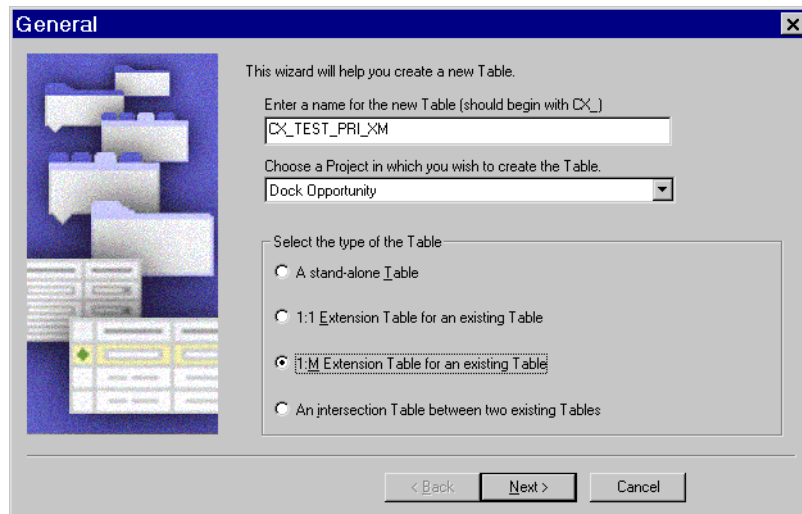
You add new dock object visibility rules for existing dock objects when you use the Docking Wizard. By creating a new dock object visibility rule, you provide access to records in existing tables to mobile users who “own” a record in the new table.

This is appropriate when the new table can act as a parent to the primary table of another, limited visibility dock object or when the new table has a foreign key to the primary table of another limited visibility dock object.

Before launching the Docking Wizard, you need to lock the project containing the customer dock object to which you want to add your new table. You also need to lock all projects containing the dock objects where your new table's parent tables reside.

To create a new table to be added to an existing dock object

- 1** Lock all necessary projects.
- 2** Select File > New Object.
The New Object dialog box appears.
- 3** Select the Table icon under the General tab.
- 4** In the first General dialog box enter the name of your new table beginning with CX_, select the project, and then select the radio button for the type of table you want.



For example, create a new one-to-many extension table of CX_TEST_PRI.

- 5** Click Next.

The Parent Table Specification dialog box appears.

- 6 Specify the parent table, for example CX_TEST_PRI.
- 7 Click Next to display the Finish dialog box with a summary of your choices.
- 8 Click Finish.

The new table is created and displayed in the Object List Editor.

To add the new table to an existing dock object

- 1 Select the Table object type in the Object Explorer.
- 2 Select the new table, for example CX_TEST_PRI_XM, in the Tables list.
- 3 Right-click and select Docking Wizard from the menu options.
The Add Table to Dock Object dialog box appears.
- 4 Select the Add the Table to an Existing Dock Object radio button.

Add Table to Dock Object

Create a new Dock Object

Add the Table to an existing Dock Object

Dock Object: DOX

Dock Object: DOX PRI
Limited

Project: [Empty]

Project: Dock Opportunity
Locked

Visibility level

Private

Enterprise

Limited

Source Column: PAR_ROW_ID

Target Table: CX_TEST_PRI

< Back Next > Cancel Help

- 5 Select an entry in the Dock Object field.

The choices are a list of all Dock Objects that contain tables to which the new table has a foreign key.

The associated locked project is displayed in the Project field.

- 6 Select an entry in the Source Column field.

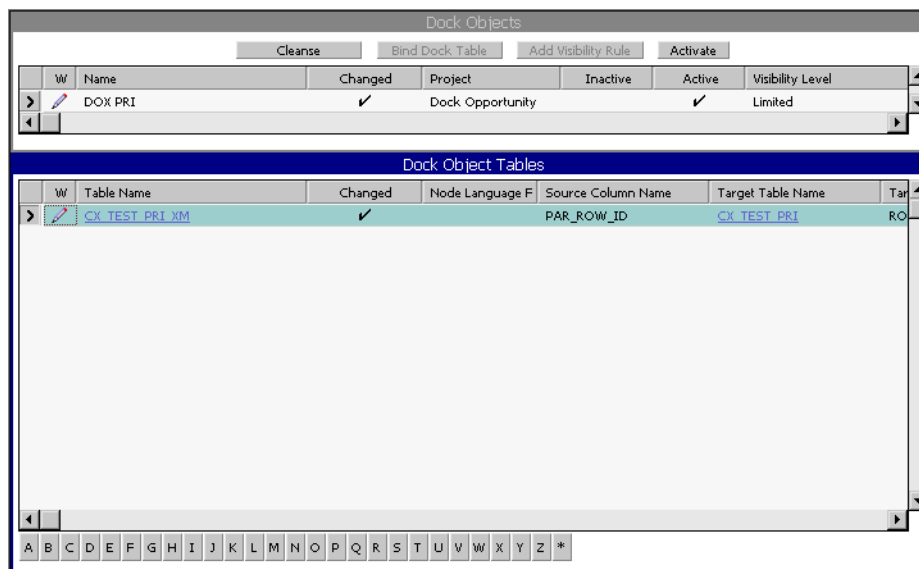
This field allows you to choose a column from the new table that is a foreign key to the parent table contained in the selected Dock Object Table. Frequently, there is only one such column, but there may be more in some cases.

NOTE: When the Source Column field entry is selected, the Target Table field is populated.

- 7 Click Next to display the Summary dialog box.

- 8 If the information displayed is correct, click Finish.

The Docking Wizard creates a Dock Object Table object based on the new table and displays it in the Object List Editor.



Verifying Dock Objects

You can verify newly created dock objects, dock object tables, and dock object visibility rules within Siebel Tools.

To verify dock objects

- 1 In the Object Explorer, select Dock Object.
- 2 Expand Dock Object, and then select Dock Object Table.
- 3 Look for the created item in the Dock Object Tables list.
- 4 Select Dock Object Visibility Rule.
- 5 Look for the created item in the Dock Object Visibility Rules list.

For more information on dock object visibility rules, see [“Dock Object Visibility Rules” on page 316](#) and *Siebel Remote and Replication Manager Administration Guide*.

Deleting and Cleansing Dock Objects

Custom dock objects (those with prefix DOX) can be deleted.

When a custom table, column, or dock object is deleted, or when a foreign key column is redefined to point to a different table, dock object integrity may be broken. The dock objects will need to be cleansed before the Docking Wizard is used again or before Siebel Remote can be used.

The Cleanse button is located on the Dock Object list applet in Siebel Tools. By clicking it, all dock objects are examined, and you are prompted to make sure the dock objects are all clean; if not, some objects will be deleted. If the projects on which you are working are not locked, you are prompted to lock them. After the process is completed, you are prompted again on what has been deleted.

Consulting Siebel Expert Services

- **Reviewing alternatives to creating new objects or new rules.** Before creating a new dock object visibility rule, you should consider all configuration alternatives. It is recommended that Expert Services be engaged to research and review any configuration alternatives that might exist. This will make sure that no dock object visibility rules are added unnecessarily.
- **Reviewing implemented changes with regards to docking.** The Docking Wizard is a new feature for Siebel 7. If for any reason you do not feel comfortable using it or analyzing its effects, please consult Expert Services.

- **Inactivating docking.** When the Docking Wizard creates new dock object visibility rules, the rules are bidirectional. For example, if you create a custom foreign key on the S_OPTY table in relation to the ROW_ID of S_ASSET then two rules are created: one for the Opportunity dock object and one for the Asset dock object.

However, the business requirement might be to dock assets as they are related to opportunities that a user receives at 100% visibility, but not to dock opportunities based on the fact that they are related to an asset that the user is receiving. Therefore, you would need to request that the new dock object visibility rule that is created in the Opportunity dock object be inactivated. Expert Services would perform this task.

You might also need a standard Siebel dock object visibility rule to be inactivated due to the volumes of data which are downloaded to mobile users. This can also be analyzed by Expert Services.

- **Activating docking.** Before adding a dock object visibility rule a team should thoroughly research and determine if an existing (but inactive) dock object visibility rule would provide the necessary functionality. If this is true, Expert Services must be engaged to activate the rule and research the performance impact.
- **Adding a new table to an existing dock object.** Currently, the Docking Wizard does not support the addition of a new table to an existing dock object. If this is a requirement, you should request assistance from Expert Services.

NOTE: The functionality that is currently supported by the Docking Wizard is to create custom dock objects and to support custom foreign key relationships between existing dock objects.

- **Changing visibility strengths.** When using the Docking Wizard to add new visibility rules, visibility strength might need to be altered because the Check Dock Object Visibility property is set to 100% and the Visibility Strength property for the dock object based on the new rule is set to 50%.

For example, if a rule is created as mentioned above to implement a custom relationship between S_OPTY and S_ASSET, the rule in the Asset dock object would check to make sure that the Opportunity dock object was received at 100%. If it was, the Asset dock object would be docked at 50%.

However, this would mean that the S_NOTE_ASSET table would not be docked because the visibility strength of this table in the Asset dock object is 100%. This means that for mobile users to receive records in S_NOTE_ASSET, they must have received the parent S_ASSET record at 100% or greater. If this scenario does not satisfy the docking requirements for asset records, Expert Services could be engaged to analyze the design and modify the visibility strengths if appropriate.

- **Implementing Non-Primary Manager Visibility.** Expert Services can activate the dock object visibility rules that assist in the implementation of Non-Primary Manager Visibility. This can have very serious performance implications and should be avoided.
- **Using routing models.** New with Siebel 7. Routing models are position- and responsibility-based models that improve the performance and synchronization time. The standard routing models all have specific rules excluded from synchronization. If you use these models and request that rules either be added or deleted from the models, Expert Services will need to be engaged.

Adding Custom Extensions to the Data Model

Dock Objects

This chapter describes objects in the Business Objects Layer of the Siebel eBusiness Applications architecture. Siebel business objects consist of:

- Business components
- Fields
- Joins
- Links

Major Business Object Types

The full set of business object types and their relationships is illustrated in [Figure 78 on page 338](#).

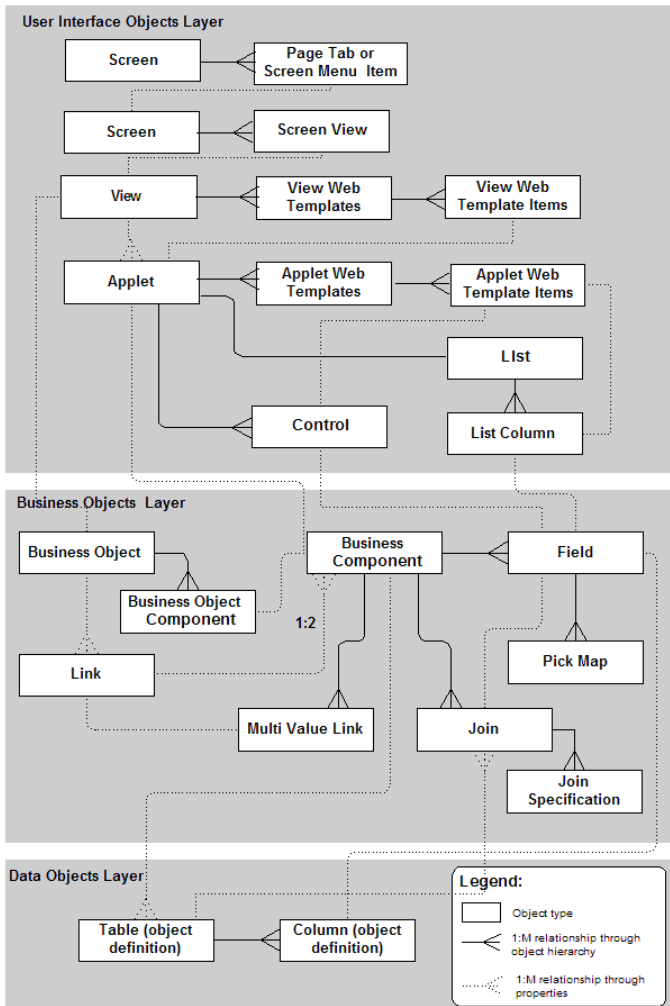


Figure 78. Details of User Interface Architecture

The following user interface object types are introduced in [Figure 78 on page 338](#):

- **Business Component.** A business component is a logical entity that associates columns from one or more tables into a single structure. Business components provide a layer of wrapping over tables, causing applets to reference business components rather than the underlying tables. This design creates convenience (all associated columns together in one bundle), developer-friendly naming, and the isolation of the developer role from the database administrator role.
- **Field.** A field object definition associates a column to a business component. This is how columns from tables are assigned to a business component and provided with meaningful names that the customer developer can change. Alternately, a field's values can be calculated from the values in other fields in the business component. Fields supply data to controls and list columns in the Web interface.
- **Business Object.** A business object implements a business model (logical database diagram), tying together a set of interrelated business components using links. The links provide the one-to-many relationships that govern how the business components interrelate in the context of this business object.

NOTE: The object type called Business Object is not to be confused with the general category called business object types. Business Object is one of the object types in the Business Objects layer. Similarly a business object, which is one kind of object definition, is not the same as the category “business object definitions.”

- **Business Object Component.** A business object component object definition is used to include a business component and, generally, a link in the business object. The link specifies how the business component is related to another business component in the context of the same business object.
- **Link.** A link implements a one-to-many relationship between business components. The Link object type makes master-detail views possible. A master-detail view displays one record of the master business component with many detail business component records corresponding to the master. A pair of links also may be used to implement a many-to-many relationship.

- **Multi-Value Link.** A multi-value link is used in the implementation of a multi-value group. A multi-value group is a user-maintainable list of detail records associated with a master record. The user invokes the list of detail records from the master record when it is displayed in a list or form applet. For example, in an applet displaying the Account business component, the user can click the Select button to the right of the Address text box to see a pop-up window displaying multiple Address records associated with the currently displayed account.
- **Join.** A join object definition creates a relationship between a business component and a table that is not the business component's base table. The join allows the business component to build fields using columns from the non-base (joined) table. The join uses a foreign key in the business component to obtain rows on a one-to-one basis from the joined table, even though the two do not necessarily have a one-to-one relationship. For example, an Account can have multiple Contacts, and each Contact identifies one Account in its foreign key. This makes it possible to generate, by means of a join between Contacts and Accounts, a list of Contacts with Account information about each.
- **Join Specification.** Join Specification is a child object type of Join that provides details about how the join is implemented within the business component.
- **Business Service.** A business service is a reusable module containing a set of methods. It provides the ability to call its C++ or script methods from customer-defined scripts and object interface logic, through the invoke-method mechanism.
- **Table.** A table object definition is the direct representation of a database table in a DBMS. It has column and index child object definitions that represent the table's columns and indexes. Table, column, and index object definitions within Siebel Tools provide a detailed picture of all of the tables, columns, and indexes in use in the DBMS.
- **Column.** A column object definition represents one column in the database table. Database columns in a database table are represented by the column object definitions that are children of the corresponding table object definition. Each column in the table has a corresponding column object definition.
- **Index.** Each index object definition identifies a physical index file in the DBMS.

- **Page Tab.** A page tab object definition associates a screen to the page tab's parent application object definition and includes it as a tab in the Tab bar.
- **Find.** A Find object definition adds a type of record to search for to the Find dialog box for the application.
- **Application Find.** An Application Find object definition associates a find to the application, adding it to the Find dialog box for that application.
- **Find Field.** A Find Field object definition adds a search field to a find.
- **Find View.** A Find View object definition adds a view to the list of possible views that can be presented in response to a find.

For more information on object types, their usage, and their properties, see *Object Types Reference*.

Usage and Configuration of Non-Licensed Objects

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

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

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

Business Components

A *business component* is a logical entity that associates columns from one or more tables into a single structure. Business components provide a layer of wrapping over tables, so that applets reference business components rather than the underlying tables. This creates convenience (all associated columns are together in one bundle), developer-friendly naming, and the separation of the developer role from the database administrator role. A business component can also have a default sort or search specification, providing records to applets in a predetermined sort order and according to a selection criterion.

When instantiated in a Siebel application, a business component is comparable to a recordset. Its definition in Siebel Tools provides the foundation for controlling how data is selected from, inserted, and updated within the tables it references.

Figure 79 illustrates business component records displayed in a list applet.

Business component records

New	Name	Site	Main Phone #	Territories	Industries	Status	URL
	Woollen Goodrich N	Boston	(617) 244-5020		animal specialties	Active	
	3Com	Santa Clara	(408) 326-5000		manufacturing indus	Gold	www.3com.com
	AMCO Communicati	Chicago, IL	(847) 491-2300		steel pipe & tubes	Active	www.amco.net
*	Alliance Program						
	Andercote Consultin	Palo Alto	(415) 389-5400		services	Active	
	Andronico	Berkeley	(510) 568-1200		groceries & related	Active	
	BG Edwards, Inc	HQ	(212) 489-1500		theatrical producers	Active	www.agedwards.com

Figure 79. Business Component Records in a List Applet

A business component contains fields. Each column whose data is included in a business component is represented with a field. The Field object type is a child of the Business Component object type. The Column object type is a child of the Table object type. The set of field child object definitions of a business component maps a corresponding set of columns into the business component. These relationships are shown in [Figure 80](#).

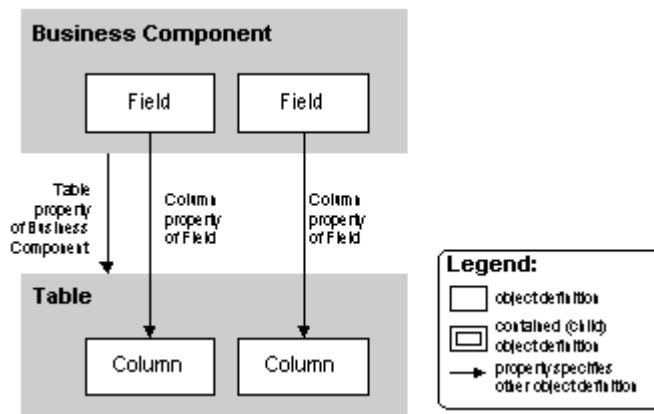


Figure 80. Business Component Field and Column Relationships

In addition to connecting a column to the business component, the Field object type also allows you to specify a meaningful name that refers to that column. Columns in tables are often cryptically named to match the names in the DBMS, whereas fields can have more meaningful and longer names than the columns they represent (75 characters long as opposed to, typically, 30 characters).

NOTE: Not all fields are representations of columns. Some fields are *calculated fields*, whose values are computed from the values in other fields. The data in calculated fields is not stored separately in the database.

Base Tables of Business Components

A *base table* of a business component is assigned to the business component to provide the most important columns for use as fields in the business component. Fields built on the base table can be edited, whereas fields built on joined tables can only be displayed.

The base table is assigned to the business component with the Table property in the Business Component's object definition. [Figure 81](#) shows an example of some fields in the Contact business component that map corresponding columns from the business component's base table, S_CONTACT.

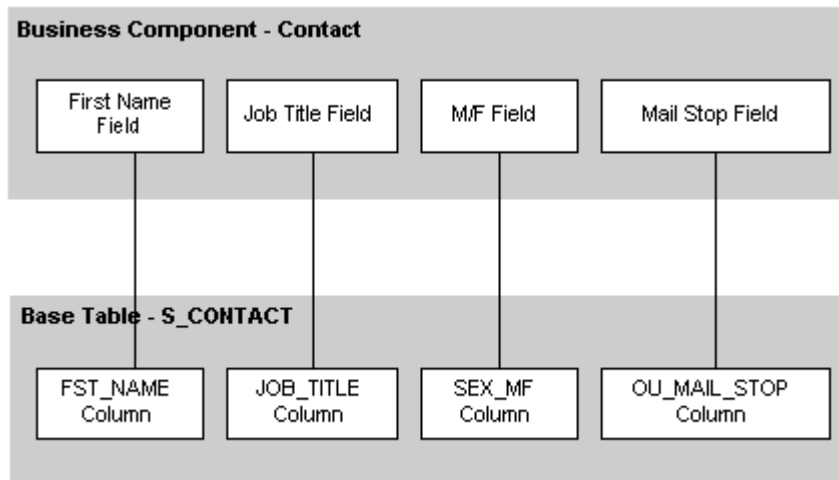


Figure 81. Examples of Fields Representing Columns

Every business component has a base table assigned to it. It is not essential that the business component include all of the columns in the base table, although typically it will include most of them. In particular, system columns in the base table such as ROW_ID, CREATED_BY and LAST_UPD_BY are automatically represented in the business component through implied fields. System columns do not require field object definitions in the business component.

Joined Tables and Extension Tables of Business Components

Not every table used by a business component is a base table. In addition to columns from the base table, columns may also be included from joined tables and extension tables. For further information on tables see chapter on Using Extension Tables and Columns.

Joined Tables

A *joined table* provides rows on a one-to-one basis to the business component as a result of a foreign key relationship between the joined table and the business component's base table. That is, for every record in the business component (which corresponds to a row in the base table) there can be a corresponding row in the joined table. However, not every record in the base table will have a record in the joined table.

The data obtained by a business component through a join (other than to an extension table) is read-only in that business component.

The use of fields from both the base table and joined tables is illustrated in [Figure 82](#).

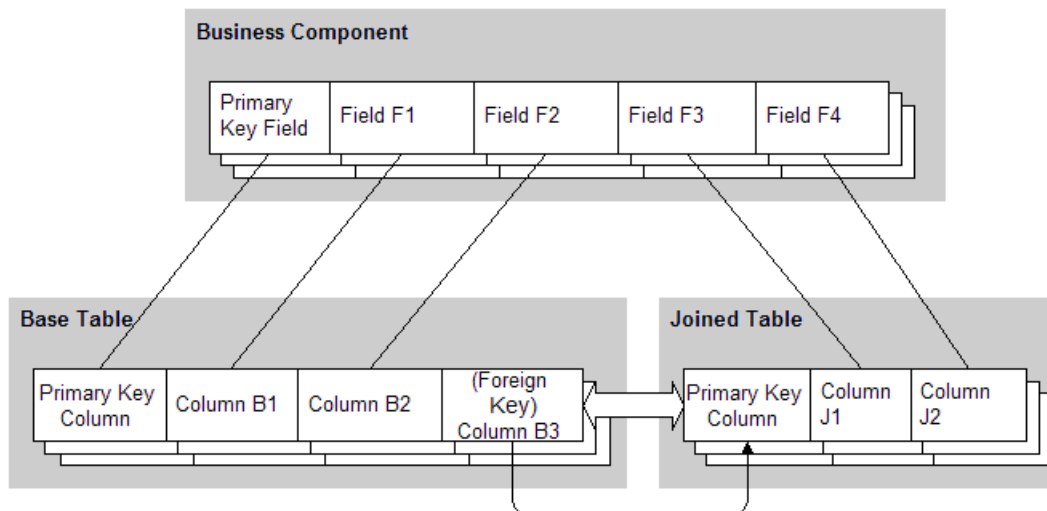


Figure 82. Fields from the Base Table and a Joined Table

Using the contact example, most contacts belongs to an account. Account information is stored in another table, S_ORG_EXT. The account is uniquely identified in each contact record (S_CONTACT row) by means of a foreign key, the Account Id field (the PR_DEPT_OU_ID column). A join uses these relationships to make account data columns available to the Contact business component for each contact. This is illustrated in [Figure 83](#).

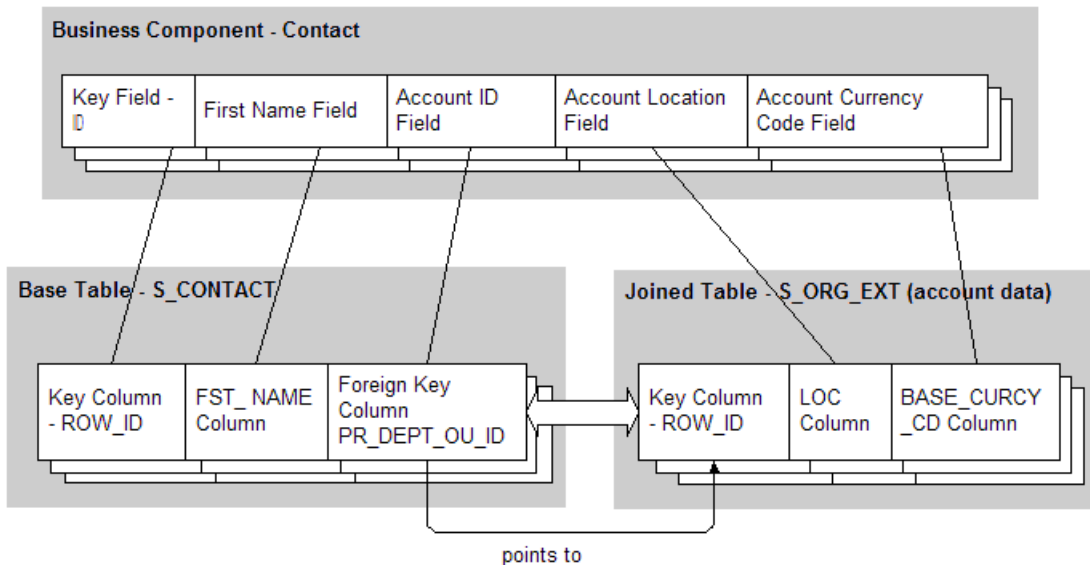


Figure 83. Fields from a Joined Table

Extension Tables

Extension tables are a special kind of joined table. Like other joined tables, extension tables provide rows on a one-to-one basis in parallel with base table rows. Extension tables are identified by the `_X` suffix in the table name, such as `S_ORG_EXT_X`, which extends `S_ORG_EXT`.

Extension tables are provided specifically to allow columns to be virtually added to a base table rather than physically added. This provides the means to expand base tables without violating DBMS or database design restrictions, and without the need to perform complicated database restructuring operations. Extension table data, unlike the data in other joined tables, can be updated in the business component.

Extension tables are discussed in greater detail in [“Extension Tables”](#) on page 196.

Sort Specification Property

The value in the Sort Specification property, if it is non-blank, is the name of a field or list of fields that imposes a sort order on the records returned to an applet that is associated with this business component. The field or fields must be child object definitions of the business component.

For example, the Account business component (as delivered in Siebel applications) has a Sort Specification property value of “Name, Location.” This indicates that account records are provided in Name (account name) order, and where multiple account records have the same Name, they are to be sorted within Name by Account Location.

Observe the following syntax considerations:

- Use commas to separate field names in a sort specification.
- To indicate that a field in the list sorts in descending order, include (DESCENDING) or (DESC) after the field name, as in “Start Date (DESCENDING).” If you do not specify a sort order, ascending order is used.
- Do not enclose the field name in square brackets, as in [Account Name]. Brackets are accepted in search specifications, but not in sort specifications.
- The sort specification expression must be 255 characters or less.

Be aware that sort specifications have the following behaviors:

- If the Sort Specification value is blank, the Siebel application returns the records in the order in which they occur in the table.
- When a check box field is included in a sort specification, there are three values that are sorted: Y, N, and NULL. If you specify that the sorting is in Descending order, the order is NULL, Y, and N.
- When a multi-value field is included in a sort specification expression in a business component, the sorting is on the initial value of the multi-value field. This makes sense only if the multi-value group uses a primary foreign key.
- A sort specification that includes a multi-value field in the expression does not sort the records in the underlying multi-value group. Instead, you create a sort specification in the detail business component of the multi-value link to do this.

- For sorting the values in a static picklist or pick applet differently than the default sorting for the underlying business component, the sort specification on the business component can be overridden with a sort specification on the picklist. The default value for the Sort Specification property in a Pick List object definition is blank, which means that the business component's sorting is to be used. If a sort specification appears in the picklist, this overrides the business component's sorting with that of the picklist.

NOTE: If a predefined query exists, it can potentially override a sort specification that has been defined as a property of the business component.

Improperly chosen sort specifications can hurt performance. This is particularly true when the sorting is on fields based on joins. Siebel Expert Services will review any custom search or sort specifications when performing a configuration review, in order to identify any potential performance issues.

Siebel applications force the sort in the All visibility mode to be on the primary key. The sort in Manager mode is on a column in the denormalized reporting relationship table. Users can still sort records after the initial query. For better performance, you should sort records after filtering for a small record set.

NOTE: Null records will always appear at the top of the record-set if a sort specification is placed on a field with null values.

Search Specification Property

If the value in the Search Specification property in a Business Component object definition is non-blank, the set of records provided to an applet using this business component is restricted. The search specification contains the names of one or more fields in the business component and various operators, combined to create a conditional expression. Records in which the value of the conditional expression evaluates to TRUE are provided to the applet for display; those records in which the expression evaluates to FALSE are excluded.

NOTE: Search specifications on child applets are not executed.

Some sample search specification expressions appear below:

```
[Type]= "COST LIST"  
[Revenue] > 5000  
[Competitor] IS NOT NULL and [Competitor] <> "N"  
[Type] = LookupValue ("TODO_TYPE", "In Store Visit")
```

Search specification expressions are built according to the following syntax rules:

- Standard comparison operators are used to compare a field to a constant, or one field to another field. These include =, < >, >, <, > =, and < =.

Example: [Revenue] > 5000

- String constants are enclosed in double quotation marks. String values are case sensitive, so the use of uppercase and lowercase letters in the search specification should exactly match that of the records you want returned.

Example: [Type] <> "COST LIST"

- The logical operators AND, OR, and NOT are used to negate or combine expressions. Case is ignored in these operators; for example, “and” is the same as “AND”).

Example: [Competitor] IS NOT NULL and [Competitor] <> "N"

- A field name in a search specification must be enclosed in square brackets.

Example: [Conflict Id] = 0

- The LIKE operator may be used to create text string comparison expressions in which a field is compared to a constant, or a field to another field, and a match on only the first several characters is required. The wildcard characters “*” and “?” are used to indicate any number of characters, and a single character, respectively.

Example: [Last Name] LIKE "Sm*"

In this example, the Last Name values of Smith, Smythe, Smallman, and so on would cause the expression to evaluate to TRUE.

- The search specification expression must be 255 characters or less.

An applet search specification cannot be used to override the search specification of the underlying business component, if the business component has one. Rather than overriding the business component's search specification, the applet's search specification is appended to that of the business component. Search specifications should appear in the business component or the applets that use it, but not both.

The search specification on an applet is converted to a WHERE clause by the data manager at runtime. When two applets based on the same business component appear in the same view, one query is generated against the database to populate both applets. Because a database select statement only supports one WHERE clause, only one of the applets should have a search specification—or if both do, they should have the same specification.

For example, the Account List Applet and the Account Entry Applet both appear in the Account List View. The record that is selected in the Account List Applet also appears in the Account Entry Applet. When you select a different row in the list or scroll through the list, the Account Entry Applet is updated to show the same record that is selected in the Account List Applet. This is made possible by the fact that both applets are populated from the same query and therefore show the same record set.

To prevent the two applets from being synchronized, they would have to be on separate business components, for example by copying the business component on which the first applet is based.

For more information on the usage of the Search Specification property of applets, see *Siebel Developer's Reference*.

NOTE: Search specifications can impact performance negatively, particularly when you include fields based on joins in the search specification. Search specifications with NOT or OR can also adversely affect performance by forcing the database to execute a full table scan.

Configuring Data-Driven Read-Only Behavior

Business components and fields can be configured as dynamically accessible, with their read-only status turned on and off depending on the value in a particular field in the current record. This is accomplished using one of the following Business Component object type user properties, depending on the requirement:

■ BC Read Only Field

Specifies a TRUE or FALSE field in the record that, when TRUE, causes the current record to become read-only.

■ Field Read Only Field: *fieldname*

Specifies a TRUE or FALSE test field and a target field in the same business component. When the TRUE/FALSE field is true, the target field becomes read-only.

NOTE: FieldName syntax works if FieldName is not a join field. If FieldName is a join field to another table, then this syntax does not prepopulate the field that uses this syntax in its Pre Default Value.

■ Parent Read Only Field

Specifies a TRUE or FALSE test business component or field combination in the parent chain (parent, grandparent, and so on) that, when TRUE, causes the target business component to become read-only.

These user properties are described individually in greater detail in the subsections that follow.

The following warnings are common to all three user properties:

- Wherever a business component or field name is specified, whether in the Name or Value property of the user property object definition, make sure that the capitalization, spelling, and use of blank spaces are correct. Also make sure that quotation marks are not present.

- These user properties do not function when used in an applet in a view where the view's Admin Mode property is set to TRUE.

Admin Mode, when TRUE, turns off all insert and update restrictions for the business components used by the view, including those specified by business component user properties. The business component Sales Rep and Personal visibility modes are ignored. Records that do not have a primary team member are also visible. However, pop-up visibility is not overridden.

NOTE: The Admin Mode flag should be set to TRUE only in a view that is part of a screen containing all administration views. Do not use the Admin Mode flag for a view in a screen that contains any non-administration views.

You can have a list view with Admin Mode set to TRUE that drills down to a detail view not marked as an administration view, while remaining in Admin Mode. This allows you to share detail views with non-administration list views.

BC Read-Only Field

This user property specifies a Boolean field that, when TRUE, causes all fields in the current record to become read-only. This also prevents the user from updating or deleting the record, but does not prevent the addition of new records to the business component. The Name and Value properties in the user property record are specified as follows:

- **Name**

Contains the literal text BC Read Only Field.

- **Value**

Contains the name of a field in the same business component as the parent object definition of the user property. This field must be a TRUE or FALSE field.

An example of the use of BC Read Only Field is the situation in which you need to prevent users from updating inactive accounts. The Inactive Account field in an account record is a TRUE or FALSE field that, when TRUE, indicates that the account is inactive. To configure dynamic read-only behavior for the Account business component based on this field, add a business component user property child object definition to the Account business component, with the following property settings:

- **Name**

BC Read-Only Field

- **Value**

Inactive Account

Field Read Only Field

This user property is similar to BC Read Only Field, in that it tests the field specified in the Value property and enforces a read-only restriction when the test field has a value of TRUE in the current record. However, unlike BC Read Only Field, the Field Read Only Field user property restricts one field in the same business component, rather than the entire business component record.

The Name and Value properties in the user property record are specified as follows:

- **Name**

Contains an expression in the following format:

Field Read Only Field: *fieldname*

For example:

Field Read Only Field: Account Status

Note that there is only a single space between the colon and the field name.

- **Value**

Contains the name of the test field. This is a TRUE or FALSE field in the same business component as the parent object definition of the user property.

One Field Read Only Field user property must be created for each field you want to make conditionally read-only.

An example of the use of Field Read Only Field is the situation in which you want to make the Competitor field in an account record read-only when the Type field has a value of Competitor. In other words, if an account record has been included because that account is a competitor, you do not want users to be specifying that account's competitors. The following procedure describes how to accomplish this.

To restrict the Competitor field in an account based on the account's type

- 1** Navigate to the Business Component object type in the Object Explorer, and then to the Account object definition in the Object List Editor.
- 2** Create a calculated Boolean field in this business component that will have a value of TRUE when the Type field has a value of Competitor.

For purposes of the example, the name of this test field can be Competitor Calc, although the name is unimportant as long as it is referenced correctly in the user property.

- 3** In the calculation property of the Competitor Calc field, enter the following value:

```
IIf([Type] = "Competitor", "Y", "N")
```

- 4** Expand the Business Component object type in the Object Explorer, and select the Business Component User Prop object type. Click the Object List Editor to make it active, and choose Edit > New Record.
- 5** Set the following values in the new Business Component User Prop object definition:

- **Name**

Field Read Only Field: Competitor

- **Value**

Competitor Calc

Parent Read Only Field

This property, like BC Read Only Field, places a read-only restriction on an entire business component, rather than a single target field. This restriction occurs when a TRUE or FALSE test field has a TRUE value. However, unlike BC Read Only Field and Field Read Only Field, this user property is used to place a restriction on a child or grandchild (and so on) business component of the business component containing the test field. In the other user properties, the read-only restriction is placed on the business component containing the test field, or on another field in the same business component.

Parent Read Only Field is used primarily to restrict the detail records in a multi-value group. It could also be used to restrict the detail records in a master-detail view, but in that case you need to make sure that the restricted business component is not also used in the context of some other business object than the intended one.

The Name and Value properties in the user property record are specified as follows:

■ **Name**

Contains the literal text Parent Read-Only Field.

■ **Value**

Contains an expression in the following format:

buscompname.fieldname

where *fieldname* is the name of the test field, that is, the TRUE or FALSE field to be evaluated, and *buscompname* is the name of the business component in which the test field is located. For example:

`Account.Inactive Account`

The business component to be conditionally restricted is the one to which you add the user property as a child object definition. The business component containing the test field must be a parent or grandparent of the restricted business component by way of a link or series of link relationships.

An example of the use of this user property is the situation where you want to disable the update of the Account Address multi-value group when the account record has a Type of Competitor. To accomplish this, you add the same calculated field as in the Field Read Only Field example, and then add a user property to the Business Address business component with the following values:

■ **Name**

Parent Read Only Field

■ **Value**

`Account.Competitor Calc`

This causes the Account Address multi-value group to be read-only when the account record is for a competitor.

NOTE: When using the Parent Read Only Field user property, the test field must have its Link Specification property set to TRUE. Otherwise the dynamic read-only functionality does not work. However, if the child record is displayed in the multi-value field in the parent business component, it is not necessary to have the Link Specification property of the field set to TRUE.

Intersection Business Components

An *intersection business component* is a business component based on an intersection table. It provides the means to display all of the combinations of data in a many-to-many relationship, instead of only one or the other one-to-many relationship of which it is composed.

Intersection tables implement many-to-many relationships. Some (such as S_OPTY_CON and S_ACCNT_POSTN) also provide intersection data through a join to one or the other master business component that uses the intersection table. Intersection data is data that resides in columns other than the two required foreign key columns in the intersection table, and is specific to the intersection of the two master business components. Intersection data columns are described in [“Intersection Data in the Intersection Table” on page 212](#).

An example of an intersection data column exposed only through a join is the ROLE_CD column in S_OPTY_CON. This column specifies the role of each contact in each opportunity, and is exposed in contact records through the S_OPTY_CON join. Exposure through a join is adequate in many circumstances, namely when the many-to-many relationship only needs to be viewed from the perspective of one or the other master business component. In this example, the purpose of exposing the intersection data column is to identify the role of each contact in a list of contacts for one opportunity.

If it were necessary to view all contacts for all opportunities in a single list, an intersection business component based on the S_OPTY_CON table would be required.

An example of an intersection business component that is exposed in the user interface is Opportunity Product. It is based on the S_REVN table, and used in the applets in the Opportunity Products screen, shown in [Figure 84](#).

The screenshot displays the 'Opportunity' form in a Siebel CRM interface. The form is divided into several sections:

- Name:** 3014 seats of eService through Mik
- Sales Method:** Standard Sales Process
- Sales Stage:** 01 - Prospecting
- Account:** Téléphone & Communications S
- Site:** Velizy/ France
- Committed:**
- Revenue:** \$3,014,576.00
- Probability %:** 50%
- Expected Value:** \$1,507,288.00
- Close Date:** 11/26/2001
- Organization:** Default Organization
- Lead Quality:** 2-Very High
- Source:** eService Seminar email: 08/20/02

Below the form is a navigation bar with tabs for 'Organizations', 'Partners', 'Products', 'Profile', 'Projects', 'Quotes', 'Revenues', 'Sales Team', 'TAS', and 'Strategic Selling'. The 'Products' tab is active, showing a table with the following columns: Product, Product Line, Auto Quote, Qty, Net Price, Revenue, and Sales Rep. The table currently contains no records.

Figure 84. Opportunity Products Screen

The list applet in this view displays all current opportunity-product combinations. When an intersection business component is used for the display and modification of data in an intersection table, it is not necessary to use an association applet to create new association rows. The user can add a new record in the list applet that displays the intersection business component, and enter or pick values in list columns that specify the two masters. In the case of the opportunity product list, an Opportunity and a Product list column are provided, and selection is made in one or the other, using the corresponding pick applet.

Intersection business components can be used to expose data from denormalized columns in intersection tables. Be sure to configure the applet controls used to display data from denormalized columns as read only. This is because denormalized columns are populated by the system, not by input through applet controls.

For more information on using intersection business components, see [“Updating Fields That Are Based on Columns in Extension Tables of Intersection Tables” on page 214](#).

Virtual Business Components

Business components based on external data are called *virtual business components*. Virtual business components are used when the business component has to obtain data from a location other than a database table in the Siebel database, but the information has to be presented in the standard Siebel user interface (applets and views). This is typically real-time information from another database, such as from the Report Encyclopedia in Actuate, or from an SAP table, although anything that can supply data in response to a SQL query is a candidate.

Virtual business components allow you to:

- Represent external data (for example, data in an SAP R/3 database) as a virtual business component within a Siebel application—the business component configuration specifies the DLL to use to access the data
- Use business services to transfer data

Virtual business components support properties such as:

- Single-value fields
- Field-level validation
- Standard business component event model (for example, PreNewRecord, PreDelete, and so on)
- Insert, delete, query, and update operations

Additional information about virtual business components:

- Applets can be based on virtual business components.
- Virtual business components can be accessed through object interfaces.
- All business component events are available for scripting.
- Virtual business components cannot be docked.
- Virtual business components can be used as stand-alone or children business components in a business object.
- Virtual business components support dynamic applet toggles. For more information about applet toggles, see [“Applet Toggles.”](#)

- Virtual business components can function as parent member of Link objects in 1:M relationships with standard business components.
 - Virtual business components generate Siebel row IDs the same way as standard business components.
 - M:M relationships involving virtual business components are not supported.
 - M:M relationships in an external system (for example, SAP) that function similarly to a 1:M relationship are supported.

For more information about using virtual business components, see *Overview: Siebel eBusiness Application Integration Volume I*.

Master-Detail Business Components

The relationships between object definitions used to implement a multi-value field that is based on a one-to-many extension table are illustrated in [Figure 85](#).

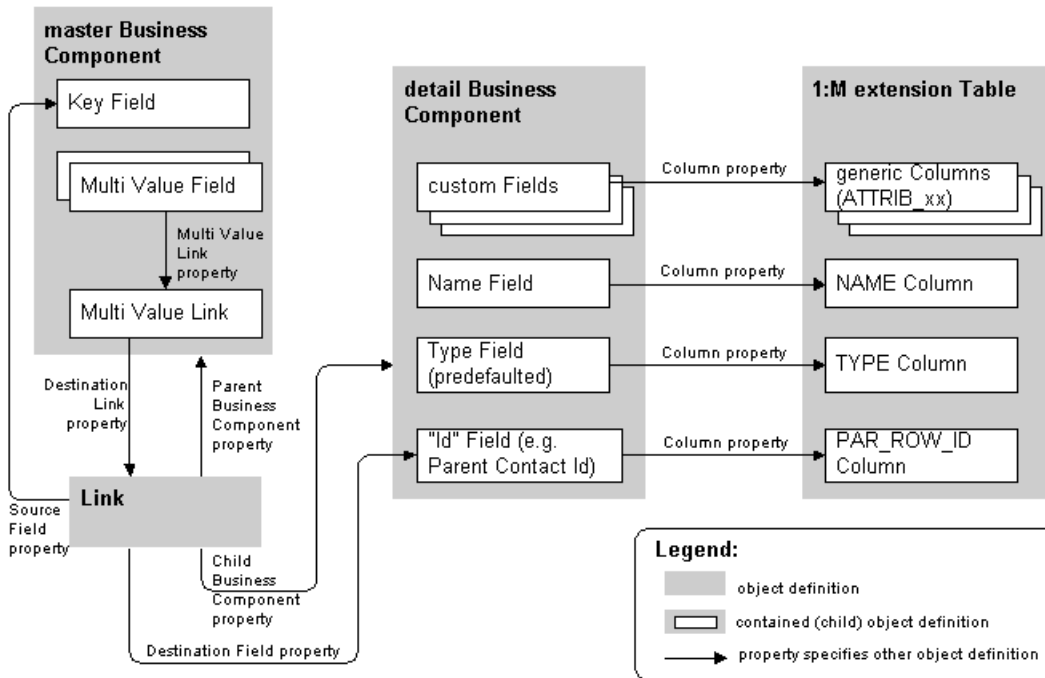


Figure 85. One-to-Many Extension Table Details

The object definitions in [Figure 85](#) are described in detail in the following section.

Master Business Components

The master business components will hold the new multi-value fields. Master business component contain the following important object definitions:

- **Key field.** This is the key field in the master business component; it is used to reference individual records. Typically, it is named Id. The Source Field property of the Link object definition points to this field. The property value may be blank because, by default, a blank Source Field value refers to the Id field.

- **Multi-value field.** The multi-value field provides access to a corresponding field value in the current record of the detail business component. The control or list column that displays the multi-value field normally will be able to invoke a multi-value group applet for display and maintenance of the detail records.
- **Multi-value link.** The multi-value link provides access to the set of records in the detail business component. One multi-value link is created for each multi-value group that is created using the one-to-many extension table.

Link

The link object definition creates the one-to-many relationship between the master and detail business components. There are no special link configuration issues related to one-to-many extension tables. For more information on the configuration of links, refer to [“Links” on page 393](#).

Detail Business Component

The detail business component represents the one-to-many extension table used by the multi-value link and multi-value group applet. Multiple custom business components can be created using the same one-to-many extension table. Each custom business component presents a different type of data for use in a different multi-value group.

The detail business component contains custom fields that represent generic (ATTRIBxx) extension columns, and hold whatever data is required for the application. For example, an Area Of Expertise business component might have a Subject Area field, a Years Of Experience field, and a Licensed field. Each field is a mapping of a different generic extension column.

The following three fields are part of the User Key (U1), which uniquely identifies a row for EIM:

- **Name field.** The name field represents the NAME column from the extension table. It provides the means for the user to enter an identifying value in each record. For example, in a Hobbies business component, the name field might be called Hobby. The user would enter the name of a hobby into each record in this field.

- **Type field.** The type field is usually named Type, and represents the TYPE column. It contains the same value for all records in one multi-value group, and distinguishes the records of that multi-value group from others. It should be set in the Predefault property to some identifying word or phrase, such as HOBBY, EXPERTISE or PRIOR JOB, and should not be exposed in the user interface.
- **Parent ID field.** The parent ID field represents the PAR_ROW_ID column. Generally it is named Parent Contact Id, Parent Account Id, or something similar. It identifies the row ID of the base table row corresponding to the parent record in the master business component. The parent ID field is specified in the Destination Field property of the Link object.

The detail business component contains one important property for use with a one-to-many extension table:

- **Search Specification.** The Search Specification property should be set to restrict the records retrieved to only those with a specific value in the Type field. This is the same value that is specified in the Pre Default Value property for that field. In this way, the only records retrieved in the business component (and, indirectly, the multi-value link and multi-value group applet) are those designated as being in this multi-value group.

NOTE: Do not define tables with names longer than 18 characters in the DB2 environment.

Fields

A field associates a column with a business component. This is how columns are assigned to a business component, and provided with meaningful names that do not require knowledge of the tables or joins of their origin.

Fields are the source of data for controls and list columns in applets. [Figure 86](#) illustrates data from fields displayed in a form applet.

The screenshot shows a form applet with several data fields. Callouts point to specific fields:

- "Displays the Name field" points to the *Name field containing "Incentive Compensation - 2500 See".
- "Displays the Account field from the" points to the Revenue field containing "\$2,625,000.00".
- "Displays the Revenue field" points to the Expected Value field containing "\$525,000.00".

Other visible fields include: Description, Sales Team (SADMIN), Opportunity Currency, Probability % (20%), Sales Stage (06 - Short List), Status, Reason, Committed (checked), Executive Priority (unchecked), *Close Date (11/30/2002), and *Created (3/1/2001 9:34:06 AM).

Figure 86. Data from Fields Displayed in a Form Applet

[Figure 87](#) illustrates data from fields displayed in a list applet.

The screenshot shows a list applet titled "Accounts" with a table of data. Callouts point to specific columns:

- "Displays the Name field for Accounts business component" points to the Name column.
- "Displays the Work Phone # field" points to the Main Phone # column.

Item	Name	Site	Main Phone #	Territories	Industries	Status	URL
	Woolen Goodrich N	Boston	(617) 244-5020		animal specialties	Active	
	3Com	Santa Clara	(408) 326-5000		manufacturing indus	Gold	www.3com.com
	AMCO Communicati	Chicago, IL	(847) 491-2300		steel pipe & tubes	Active	www.amco.net
*	Alliance Program						
	Andercote Consultin	Palo Alto	(415) 389-5400		services	Active	
	Andronico	Berkeley	(510) 568-1200		groceries & related	Active	
	BG Edwards, Inc	HQ	(212) 489-1500		theatrical producers	Active	www.agedwards.com

Figure 87. Fields Displayed in a List Applet

As you can see from [Figure 86](#) and [Figure 87](#), controls in a form applet and list columns in a list applet obtain their data from fields in the business component used by the applet. The Field property setting in a Control or List Column object definition specifies the field. The Business Component property in the applet specifies the business component. These property relationships are illustrated in [Figure 88](#).

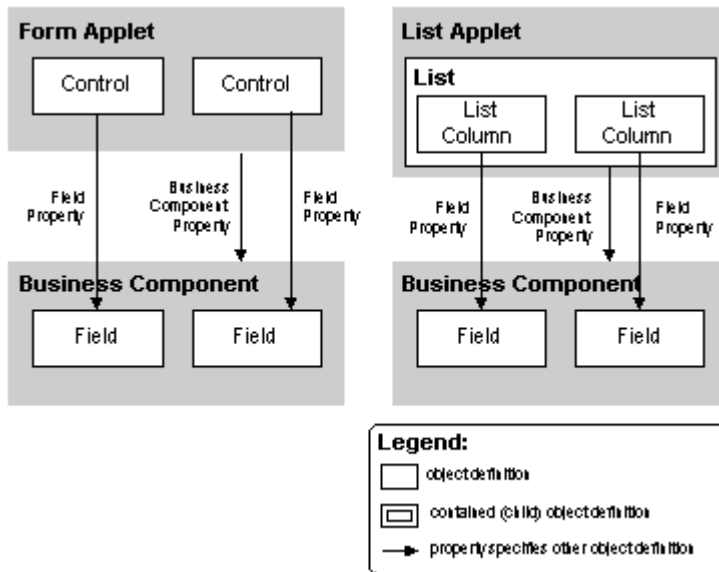


Figure 88. Field Property Relationships

Field is a child object type of Business Component. A field represents information from a database column obtained through the corresponding column object definition. Columns may be from the base table, extension tables, and joined tables of the business component. Alternately, a field may be a *calculated field* whose values are derived from the values in other fields in the business component, but not stored in the database.

In the field object definition (for other than calculated fields), the column and Join properties together specify the table and column from which the field's data is obtained. The Join property, if blank, indicates that the column is obtained from the business component's base table. If it is non-blank, the Join property identifies the join object definition that supplies data from an extension table or other joined table. Based on the Join property, the table supplying the field's data is identified. The Column property identifies the column to use within the specified table. These relationships are illustrated in Figure 89.

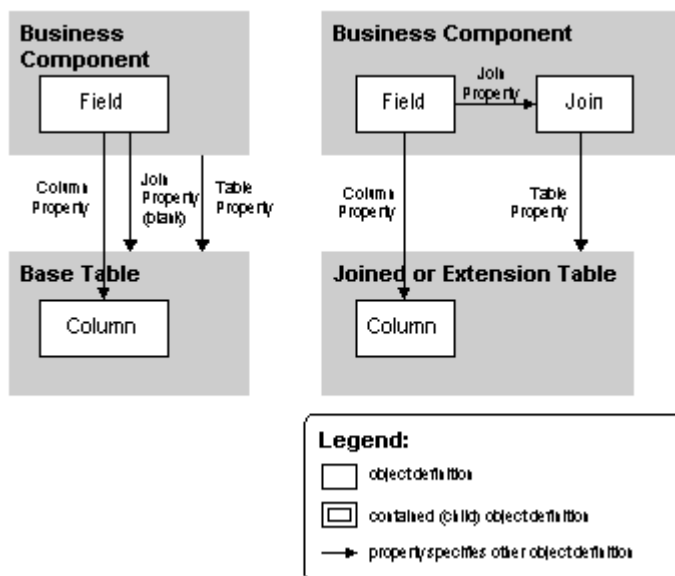


Figure 89. Field Relationship Details

NOTE: You should not map multiple fields to the same column in a table. This can lead to error messages when updating the data. The SQL query fails because it tries to access the same column twice in the same query.

System Fields

System fields are provided in all business components in standard Siebel eBusiness Applications. These fields represent the data from system columns, which are described in “[System Columns](#)” on page 222.

[Table 32](#) identifies the correspondences between system fields and system columns.

Table 32. System Fields and Their System Columns

System Field Name	System Column Name	Description
Id (or blank)	ROW_ID	Primary key for the table.
Created	CREATED	Creation date and time of the row.
Created By	CREATED_BY	User logon ID of the person who created the row.
Updated	LAST_UPD	Date of last update of the row. Updated (system field) is updated only when the row with the Updated column in it is changed.
Updated By	LAST_UPD_BY	User logon ID of the person who last updated the row.

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

NOTE: Do not change system fields, for example by renaming them. Changing Siebel system fields is not supported.

Calculated Fields

Calculated fields have a Calculated property of TRUE and a non-blank Calculated Value property. Calculated fields obtain their values from other fields in the same business component, or from the master business component in an active link in which the current business component is the detail.

NOTE: Calculated fields are not automatically refreshed when a related field value changes; they are refreshed only after committing the record. To have them refresh immediately after the fields have been changed the Immediate Post Changes property of the field needs to be set to TRUE.

The Calculated Value property contains an expression built from field names, standard functions, and string, numeric and logical operators. For example, the Full Name field in the Contact business component has the following Calculated Value property setting:

```
IIf (Language () = "JPN", [Last Name] + ' ' + [First Name],  
    [First Name] + ' ' + [Last Name])
```

The meaning of this expression is as follows: if the active client language setting is Japanese, construct the Full Name from the Last Name, a blank space, and then the First Name. Otherwise, construct the Full Name from the First Name, a blank space, and then the Last Name.

NOTE: A calculated field cannot reference itself in the Calculated Value property. For example, you cannot use [Last Name] in a calculation expression for the Last Name field.

For information on the construction of calculated field expressions for the Calculated Value property, see *Siebel Developer's Reference*.

NOTE: Queries on calculated fields are not supported if the Cache Data property of the business component is set to TRUE.

If you need to remove a calculated field, use the following procedure.

To remove a calculated field

- 1** Delete the calculated fields from the desired list applet or set the Inactive property of the List columns objects at the Applet Level to TRUE.
- 2** Navigate to the desired Business Component > Field objects level and set the Inactive Property for the calculated field to TRUE.
- 3** Set the Force Active property to FALSE.
- 4** Compile the relevant projects.

Field Data Types

The Type property specifies the data type for the field. Field data types are used to identify the type of data retrieved from and sent to the database server.

NOTE: Calculated fields are not automatically refreshed when a related field value changes; they are refreshed only after committing the record. To have them refresh immediately after the fields have been changed the Immediate Post Changes property of the field needs to be set to TRUE.

These data types are not mapped to the physical data types defined by the database. The data type of the field is generally more specific than the data type of the underlying column (as identified in the Physical Type property of the column). For example, both DTYPE_NUMBER (decimal) and DTYPE_INTEGER field data types have the Number physical data type in the column.

It is not recommended to map a field to a different table column type, for example a DTYPE_NUMBER field mapping to a table column of type Varchar.

Just as the data type of the underlying column restricts the set of field data types that will work correctly, the data type of the field restricts the set of correctly functioning format options in the control or list column that displays it.

Most formatting is defaulted from the Microsoft Windows Control Panel. Overriding the default format in the repository is possible but might lead to confusion. For example, overriding a number format to show more or fewer decimal places would be useful, but overriding a date format to DD/MM/YY would be confusing to a user who has set the date format to MM/DD/YY in the Control Panel.

NOTE: Multi-value fields (fields with a Multi Valued property setting of TRUE) have a blank Type property, because the data type of the field is specified in the detail business component that populates it.

All field data types are prefaced with DTYPE_. [Table 33](#) describes the Siebel field data types.

Table 33. Field Data Types

Field Data Type	Physical Type	Max. Length	Description
DTYPE_BOOL	Character	1	Refers to data stored as Y or N, often displayed as TRUE or FALSE and checked or unchecked.
DTYPE_CURRENCY	Number	22	<p>Refers to the data as currency.</p> <p>You can control the appearance of currency values on a screen through the Windows Control Panel, or you can specify an explicit format mask in the Display Format property by using the following symbols:</p> <ul style="list-style-type: none"> ■ Dollar sign (\$). Specifies the position for the currency symbol. ■ Trailing period (.). Specifies the default precision for the currency. ■ All valid symbols described for DTYPE_NUMBER.
DTYPE_DATE	Date	7	<p>Refers to the data as a date. When the date is returned, any additional information (for example, time) is ignored. You can set the appearance of date values through the Windows Control Panel, or you can specify an explicit date format using the following symbols:</p> <ul style="list-style-type: none"> ■ YY. Two-digit year without a leading zero. ■ Y. Two-digit year with a leading zero. ■ YYYY. Four-digit year without a leading zero. ■ YYY. Four-digit year with a leading zero. ■ MM. Month without a leading zero. ■ M. Month with a leading zero. ■ DD. Day without a leading zero. ■ D. Day with a leading zero. ■ Slash (/). Position of the date separator (you specify the character in the Windows Control Panel).

Table 33. Field Data Types

Field Data Type	Physical Type	Max. Length	Description
DTYPE_DATETIME	Date Time	7	<p>Refers to the data as a date and time. You can set the appearance of time and date values through the Windows Control Panel, or you can specify an explicit date format using a combination of the symbols for DTYPE_DATE and DTYPE_TIME.</p> <p>Alternatively, you can use one of the following three properties:</p> <ul style="list-style-type: none"> ■ Date. Displays only the date portion of the value, using the format specified in the Windows Control Panel. ■ Time. Displays only the time portion of the value, using the format specified in the Windows Control PanelTimeNoSec. Displays only the hour-and-minute portion of the value, using the format specified in the Windows Control Panel. ■ TimeNoSec. Displays only the hour-and-minute portion of the value, using the format specified in the Windows Control Panel.
DTYPE_UTCDATETIME	UTC Date Time	30	<p>Indicates that the corresponding field represents date information, with both a date and a time component, that will be stored in the database in UTC time (UTC is the equivalent of Greenwich Mean Time without any adjustments for daylight savings time). Fields of this type should correspond to database columns of type <i>U</i>, and the display values for these fields will be converted to/from UTC based on the default time zone specified in the user's preferences.</p>
DTYPE_ID	Varchar	15	<p>Refers to the data as the primary key automatically generated by the application.</p> <p>Fields mapped to extension columns of physical type Varchar(15) will automatically default to data type DTYPE_ID.</p>
DTYPE_INTEGER	Number	22*	<p>Refers to data as whole numbers ranging in value from - 2147483648 to 2147483647.</p>

Table 33. Field Data Types

Field Data Type	Physical Type	Max. Length	Description
DTYPE_NOTE	Long	16 KB	<p>Refers to the data as a long string less than or equal to 16 KB (16383 bytes); the default, if the length is not explicitly defined, is 16 KB. When used with the Pop-up Edit property in a control or list column, this data type is used to indicate to the user interface that a multiline edit box should be used.</p> <p>Users cannot query on fields of type DTYPE_NOTE.</p>
DTYPE_NUMBER	Number	22	<p>Refers to the data as a number.</p> <p>You can control the appearance of numeric values through the Windows Control Panel, or you can specify an explicit format mask using the following symbols:</p> <ul style="list-style-type: none"> ■ Zero (0). Specifies the position of a mandatory digit. ■ Pound sign (#). Specifies the position of an optional digit. ■ Comma (.). Specifies the position of the thousands separator (you specify the character in the Windows Control Panel). ■ Period (.). Specifies the position of the decimal separator (you specify the character in the Windows Control Panel). ■ Trailing period (.). Specifies default display precision. ■ Plus sign (+). Specifies the position and appearance of negative value indicator (plus sign if positive, minus sign if negative). ■ Minus sign (-). Same as plus sign.
DTYPE_PHONE	Number	40	<p>Refers to the data as a phone number. The DisplayFormat property is ignored for values of this type.</p>
DTYPE_TEXT	Varchar	2 KB	<p>Refers to the data as a string less than or equal to 2000 bytes; the default is 255. The DisplayFormat property is ignored for values of this type.</p> <p>You can use ForceCase = "Upper" or ForceCase = "Lower" to force the text to all uppercase or all lowercase after the end user tabs out of the field. You can use ForceCase = "FirstUpper" to force the first letter of each word to uppercase after the user steps off the record. Otherwise, the text is in mixed case as the user entered it.</p>

Table 33. Field Data Types

Field Data Type	Physical Type	Max. Length	Description
DTYPE_TIME	Time	7	<p>Refers to the data as a time.</p> <p>When the time is retrieved, any additional information (such as date) is ignored. You can set the appearance of time values through the Windows Control Panel, or you can specify an explicit time format using the following symbols:</p> <ul style="list-style-type: none"> ■ HH. Hour (based on 24-hour clock) without a leading zero. ■ H. Hour (based on 24-hour clock) with a leading zero. ■ hh. Hour (based on 12-hour clock) without a leading zero. ■ h. Hour (based on 12-hour clock) with a leading zero. ■ mm. Minute without a leading zero. ■ m. Minute with a leading zero. ■ ss. Second without a leading zero. ■ s. Second with a leading zero. ■ Colon (:). The position of the time separator (you specify the character in the Windows Control Panel).

NOTE: You cannot add long (16 KB limit) columns to Siebel base tables.

Sequence Fields

Situations can occur in which you need to create a field that provides sequential numbering for the parent business component. For example, you may need to number line items in an Order or products in an Opportunity. Sequential numbering is not automatically provided in any system columns in standard tables in Siebel applications. However, you can configure a sequence field in a detail business component by adding a business component user property called Sequence Field and creating a sequence business component with a special business component class called CSSSequence.

The details of configuration of a sequence field appear in [Figure 90](#).

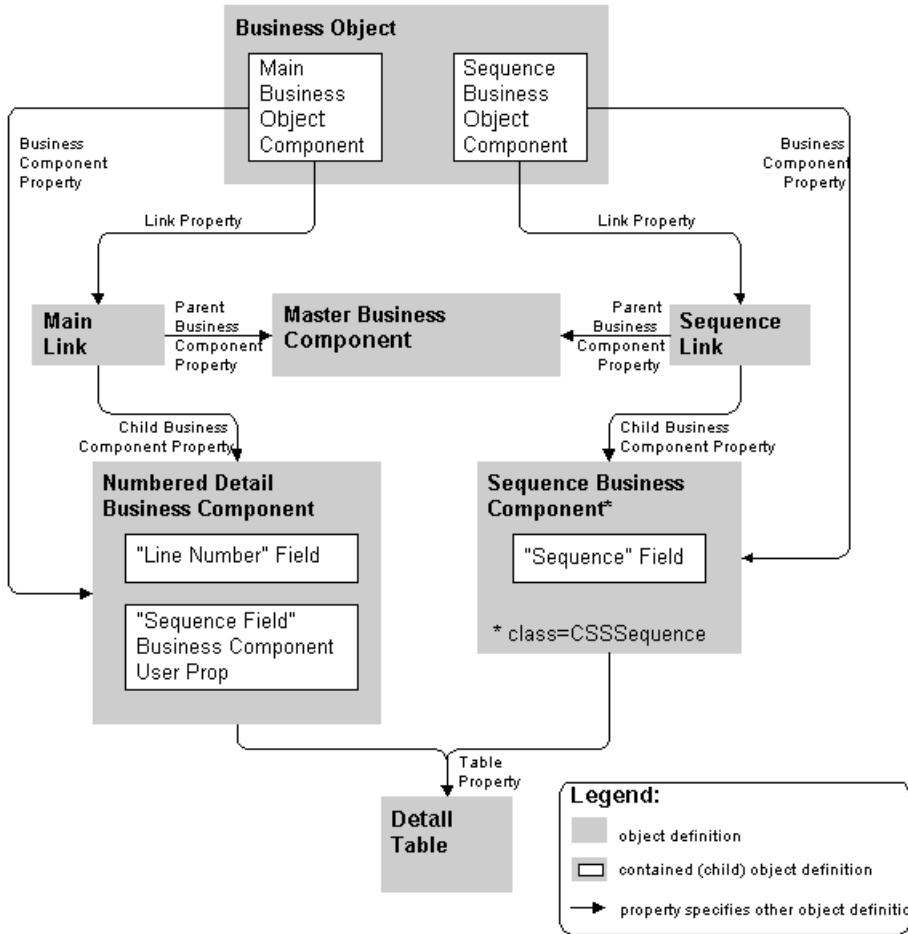


Figure 90. Sequence Field Configuration Details

The roles of the object definitions in [Figure 90 on page 374](#) are as follows:

- **Master business component.** In the master-detail relationship in which the detail records are to be numbered, this is the business component that holds master records. For example, the Opportunity business component is the master in the master-detail relationship with Opportunity Product.
- **Numbered detail business component.** In the master-detail relationship, this is the business component that holds detail records. For example, the Opportunity Product business component is the detail in the master-detail relationship with Opportunity. The numbered detail business component has the following important child object definitions:
 - **Line Number field.** This field, named Line Number, is a field of type DTYPE_NUMBER that holds the resulting sequence value.
 - **Business component user.** A business component user property object definition named Sequence Field needs to be present, with the Value property set to “Line Number.”
- **Sequence business component.** This business component is named *xx.Line Number (Sequence)*, where *xx* is the name of the numbered detail business component. It has a specialized class of CSSSequence, and the following two fields:
 - **Sequence field.** This field, named Sequence, is of type DTYPE_NUMBER.
 - **Foreign key field.** This field is a foreign key field based on a foreign key column in the detail table. The foreign key column points to the primary key of the base table of the master business component, and may be used to specify the link between the master and sequence business components.
- **Detail table.** The detail table is the base table for both the numbered detail and sequence business components.
- **Links.** One link provides the master-detail relationship between the master and numbered detail business components. The other link provides the master-detail relationship between the master and sequence business components. The link to the numbered detail business component is usually pre-existing, such as Opportunity or Opportunity Product. The link to the sequence business component is usually added by the developer, except when the sequence configuration is included in standard Siebel applications. Opportunity or Opportunity Product.Line Number (Sequence) is an example of a link to a sequence business component.

- **Business Object.** The second link is included in the same business object that holds the first link.

For an example of sequence field configuration and procedures see [Chapter 8, “Defining Business Objects and Business Components.”](#)

Examples of sequence field configuration in standard Siebel applications can be viewed in Siebel Tools. For example, examine the Opportunity Product and Opportunity Product.Line Number (Sequence) business components, which are the numbered detail and sequence business components, respectively.

To add a sequence field to a business component that does not currently have one

NOTE: Before you begin the procedure, check to see if the class of the detail business component is CSSBCBase. If it is not, you need to contact Siebel Technical Services before going any further.

- 1 Verify that the business component to which you want to add a sequence field is the detail business component in a master-detail relationship. This is the numbered detail business component.

NOTE: The numbering of detail records will start from 1 within each master record.

- 2 Add a child field object definition to the numbered detail business component. Set the Name property value to Line Number, the Type to DTYPE_NUMBER and the column to a numeric extension column such as ATTRIB_14.
- 3 Add a child business component user prop object definition to the numbered detail business component. Set the Name property value to Sequence Field, and the Value property to Line Number.
- 4 Create a business component. Set the Class property to CSSSequence, the table to the name of the base table of the numbered detail business component, and the Name to *xx*.Line Number (Sequence), where *xx* is the name of the numbered detail business component. This is the sequence business component.
- 5 Set the Sort Spec of the *xx*.Line Number (Sequence) business component to Sequence (DESCENDING).

- 6** Add a child field object definition to the sequence business component. Specify a Name property value of Sequence, and a column value the same as the extension column specified for the Line Number field in the numbered detail business component.
- 7** Add a child field object definition to the sequence business component. This is the foreign key field that establishes the master-detail relationship to the master business component. The Column property should be set to the same column as the corresponding field in the numbered detail business component.
- 8** Create a link object definition that establishes a master-detail relationship between the master and sequence business components.
- 9** Create a Business Object Component child object definition of the business object or business objects that use the existing link between the master and numbered detail business components. Specify the new link and the sequence business component in the Link and Business Component properties, respectively.
- 10** Expose the Line Number field in applets that display records from the numbered detail business component.

Joins

A Join object definition creates a relationship between a business component and a table other than its base table. The join allows the business component to use columns from that table. The join uses a foreign key in the business component to obtain rows on a one-to-one basis from the joined table, even though the two do not necessarily have a one-to-one relationship. Figure 91 shows the Contacts list displaying two list columns obtained from a join.

The Contacts list displays records from the Contacts business component.

This list column displays the NAME column from a join to the S_ORG_EXT table.

This list column displays the LOC column from the same join.

New	Last Name	First Name	Mr/Ms	Work Phone #	Job Title	Email	Account
*	Aamot	Gina	Mr.	+35555514552	Dbm	Gina_Aamot@heineken.nl	AA Consulting
*	Abanilla	David	Mr.	(309) 555-3347	Proj Leader-Architecture	David_Abanilla@davesworld.net	AA Consulting
*	Abate	Laura	Dr.	(420) 387-7729	Executive Director	Laura_Abate@Dresdner-Bank.com	AA Consulting
*	Abdallah	Trey	Mr.	(770) 555-4565	Specifications & Order Processing	Trey_Abdallah@southwire.com	AA Consulting
*	Abel	Robert	Mr.	(860) 555-5186	Assistant Director	Robert_Abel@thehartford.com	AA Consulting
*	Abelman	Brent	Mr	+44 55502406000	Proj Mgr	Brent_Abelman@scoffield.com	AV Inc.
*	Abi Farah	Mark	Mr.	+44 55573154806	Bus Support Mgr	Mark_AbiFarah@Nike.com	AV Inc.

Figure 91. List Columns Obtained from a Join

A Contact business component record represents a contact person at an account. Therefore, one account record has one or more contact records, meaning that there is a one-to-many (master-detail) relationship between the tables holding account and contact information. A detail record (or row) in a master-detail relationship always has one master record (or row), as illustrated in Figure 92.

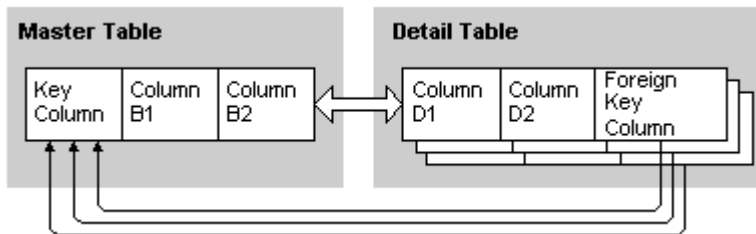


Figure 92. Master-Detail Relationship in a Join

The master-detail relationship is implemented with a foreign key column in the detail table. Multiple rows in the detail table have the same foreign key value pointing back to the same row in the master table.

Returning to the accounts and contacts example, you can look at accounts (S_ORG_EXT table rows) from the perspective of contacts (S_CONTACT table rows). Each detail table row (S_CONTACT) has exactly one master table row (S_ORG_EXT) in the one-to-many relationship. The one account row for each contact row makes it possible to treat account rows as if they were appended onto the ends of the contact rows. This provides account information about each contact's account, along with the other contact information.

NOTE: Contact may have 1 or 0 Accounts.

A business component whose base table is a detail table in a master-detail relationship can include columns from the master table as fields. This is the principle behind a join.

Figure 93 illustrates the set of rows resulting from a join between the Contacts business component and S_ORG_EXT (accounts) table.

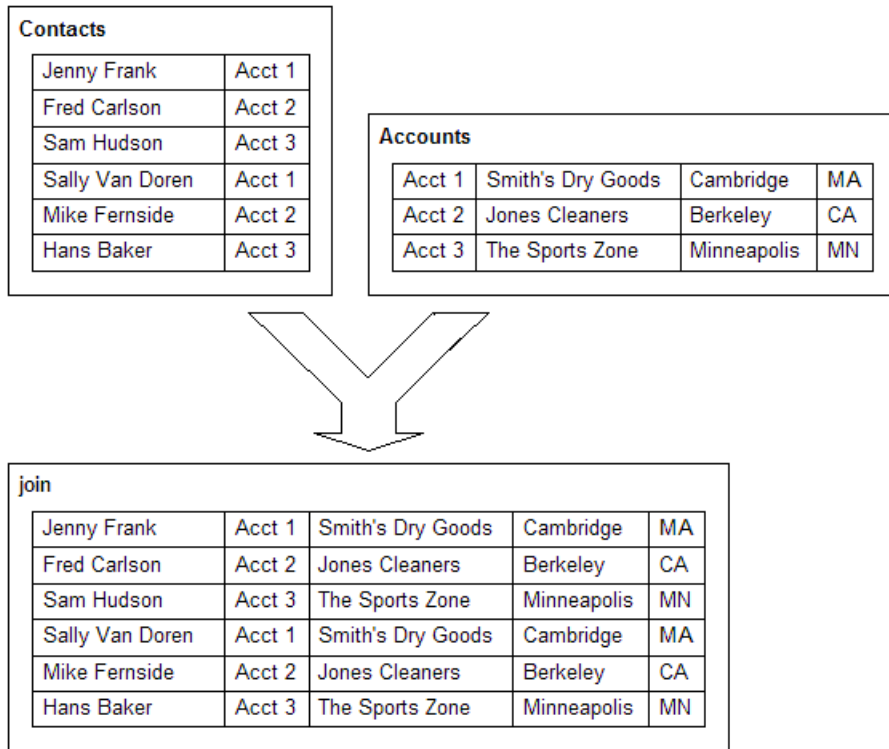


Figure 93. Set of Rows Resulting from a Join

In the diagram, the account number (Acct 1) in Contacts is the foreign key.

A join is always one-to-one and it is always between a business component and a table. Once a join is created, you can create additional fields in the business component based on columns in the joined table. In the diagram, the account name, city and state are fields that can be added to the Contact business component because of this join.

NOTE: It is possible to base a join on a joined field. It is possible to use a joined field as Source Field on the join specification. This is important if, for example, you need to join in grandparent data through the parent id field on the parent business component.

How a Join Is Constructed

The object definition relationships in a join are illustrated in [Figure 94](#).

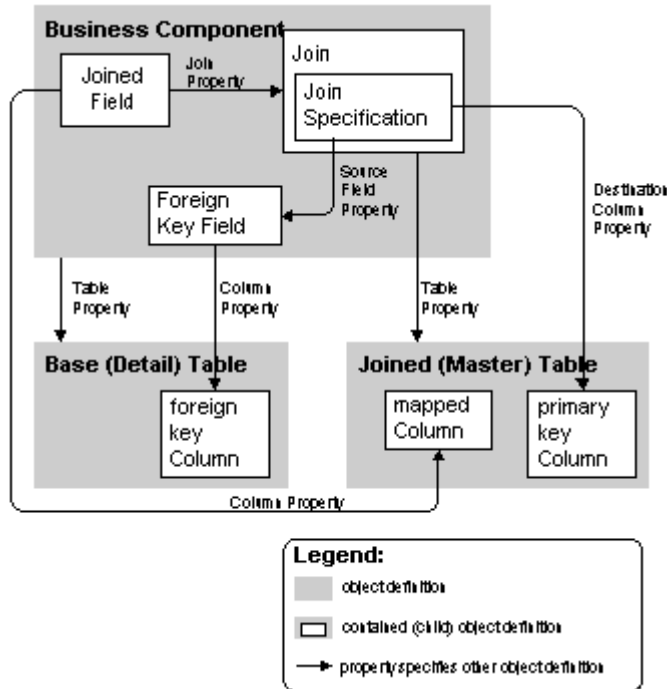


Figure 94. Join Relationships

The roles of the object definitions in the diagram are summarized as follows:

- **Business Component object type.** The business component is the parent object definition of the join. Because of the join, fields in the business component (called joined fields) can represent columns from the joined table.

- **Joined field.** A joined field in the business component represents a column from a table other than the business component's base table. Therefore, a joined field must obtain its values through a join. A joined field has the name of the join in its Join property. Together the Join property and Column property identify the column and how to access it. When creating a joined field in a business component, you can change the Type property from the default DTYPE_TEXT to a more appropriate type. For example, if you are joining a table column that contains phone numbers, you can change the Type field to DTYPE_PHONE.
- **Join object type.** Join is a child object type of the Business Component object type. The Join object definition uniquely identifies a join relationship for the parent business component and provides the name of the destination (joined) table. The join object definition identifies the joined table in the Table property. The name of the base table is already known to the business component. Typically, a join object definition is given the same name as the joined table.
- **Join Specification object type.** The join specification object definition is a child of the join object definition. It identifies the foreign key field in the business component and the primary key column in the joined table (that the foreign key points to).

The Source Field property identifies the foreign key field in the business component. If left blank, the Source Field is the Id field, indicating a one-to-one relationship between the business component and the joined table. Occasionally, a system field such as Created By or Updated By may be specified as the foreign key field in the Source Field property.

The Destination Column property identifies the primary key column in the joined table. A non-blank Destination Column property value is required if the join occurs on a column other than ROW_ID. A blank value in the Destination Column property means that the destination column is ROW_ID, which is typically the primary key in tables in Siebel applications.

NOTE: In rare circumstances, there can be multiple join specifications in a single join. For example, the Sub Campaign business component has a join to the S_LANG table with two join specifications. In such cases the source fields in the join specifications should be based upon the same table.

- **Join Constraints.** A join constraint is a constant-valued search spec applied to a column during a join. It is for use with outer joins.

NOTE: Set the Outer Join Flag to TRUE if you want to retrieve all the records in the business component even when the joined fields are empty.

- **Foreign key (source) field and foreign key column.** The foreign key field is identified in the Source Field property of the join specification. It represents a foreign key column in the base table, pointing to rows in a particular table used in joins. For example, in the Contact business component, the foreign key field to the join on accounts data is the Account Id field, which represents the PR_DEPT_OU_ID column in the base table.
- **Joined table.** The joined table is the master table in the master-detail relationship. It provides columns to the business component through the join. The joined table is identified in the Table property of the Join object definition.

NOTE: When configuring a recursive or self join, the Alias name of the joins must be different than the Table Name. Using the same name will result in the following error message: “Table ‘T1’ requires a unique correlation name.” This error is often due to a faulty recursive or self join definition.

- **Primary key (destination) column.** The join specification identifies the primary key column in the joined table (in the Destination Column property). Every standard table in standard Siebel applications has a ROW_ID column that uniquely identifies rows in the table. ROW_ID is the destination in most joins.
- **Mapped column.** Columns in the joined table are available for use in fields in the business component.

Using a Predefault Value for a Join Field

Since a join field cannot be updated, you cannot use a predefault value in the regular way as a default field value if nothing is specified when a record is inserted. You can use a predefault value for a join field to show the join field value immediately as the new record is being inserted.

To use a predefault value for a join field

The following procedure uses the Opportunity Product business component as an example.

- 1** Define a join to S_OPTY in the Opportunity Product business component.
- 2** Define two new fields based on the join to show Opportunity Sales Stage and Name.
- 3** Add the two fields to the Opportunity Product applet.
- 4** Compile and test using the standard Opportunities—Products view.
 - a** Add a new Product for an Opportunity.
 - b** Note that the join fields are not populated until you requery the applet.
(However, the source field—Oppty Id—for the join is populated.)
- 5** Set the Predefault properties of the new fields to Parent: 'ParentBusinessComponent.JoinedField'.

For example, predefault Opportunity Name with Parent: 'Opportunity.Name' and predefault Opportunity Sales Stage with Parent: 'Opportunity.Sales Stage'.

- 6** Set the Link Specification property of the joined fields (Name and Sales Stage) in the parent business component to TRUE.
- 7** Compile and then add a new product for an Opportunity.

The join fields are populated immediately, and you do not need to requery the applet to see them.

Party Business Components and Joins: Party Extension Tables

Party business components consist of business components that represent all types of Siebel person and organization entities and reference the S_PARTY table. The following are examples of party business components:

- Account
- Contact
- User
- Organization
- Employee
- Position
- Household
- Access Group
- User List

S_PARTY has many extension tables, including S_ORG_EXT, S_CONTACT, S_POSTN, S_USER, and so on. Party business components store their main data in these extension tables.

You can bring party data into non-party and party business components for display in applets. (Party data refers to data stored in a S_PARTY extension table such as S_CONTACT (contacts) or S_ORG_EXT (accounts).)

NOTE: For a description of the S_PARTY data model, see [Chapter 5, “Data Objects Layer.”](#)

Mapping Fields in Party Business Components

There is an implicit join available for each extension table for a base table. Party business components store their main data in S_PARTY extension tables (Figure 95). The main fields are mapped using the implicit join for the extension table (Figure 96 on page 388).

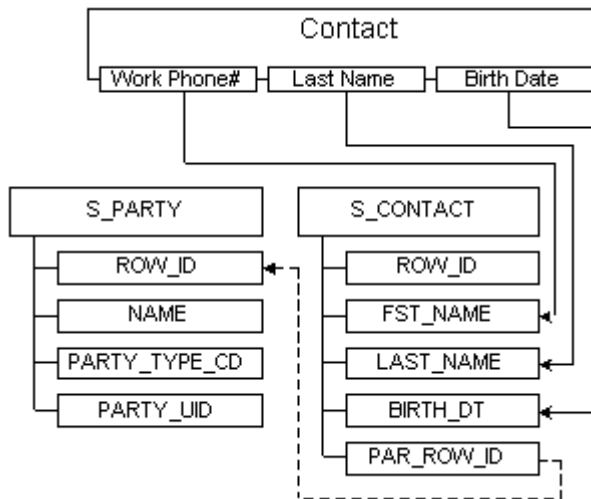


Figure 95. Mapping Extension Tables to S_PARTY

Business Components						
W	Name	Changed	Project	Cache Data	Class	Data Source
>	Contact	✓	Contact		CSSBCLUser	

Single Value Fields				
W	Name	Changed	Join	Column
	Last Close Date		S_CONTACT	DEDUP_DATACLNSD_DT
	Last Name		S_CONTACT	LAST_NAME
>	Last Name, First Name			
	Last Update - SDQ		S_CONTACT	LAST_UPD
	Login Name		S_USER	LOGIN
	Login Password		S_CONTACT_X	ATTRIB_03

Figure 96. Mapping Fields in Party Business Components Using Implicit Joins

Bringing Party Data into a Non-Party Business Component

For example, you can bring account data from the S_ORG_EXT table into the Opportunity business component to display in an applet, as shown in [Figure 97](#).

Opportunity:

Opportunities							
New		Save		1 - 1 of 1			
New	Priority Flag	Name	Account	Primary	Revenue	Sales Stage	Clos
		Call Center 999	J.F. Davison Fund	SADMIN	\$0.00		7/21/2

Figure 97. Bringing Party Data into a Non-Party Business Component

Join Definition

The join definition references the extension table storing the data of interest (Figure 98).

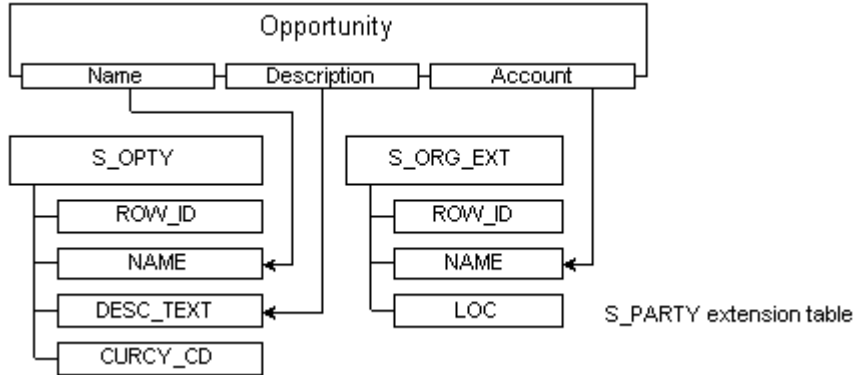


Figure 98. Join Definition

Join Specification Definition

Source Field references the relevant foreign key column to the joined table. Destination Column references the PAR_ROW_ID column in the joined table (Figure 99).

Joins						
W	Table	Changed	Alias	Outer Join Flag	Inactive	Comments
	S_LST_OF_VAL		S_LST_OF_VAL	✓	✓	obsolete in S20
▶	S_ORG_EXT		S_ORG_EXT	✓		
	S_ORG_EXT_T		S_ORG_EXT_T	✓		
	S_POSTN		S_POSTN	✓		MSUNDARA:CF
	S_USER		S_USER	✓		MSUNDARA:CF

Join Specifications					
W	Name	Changed	Destination Column	Inactive	Source Field
▶	Account Id		PAR_ROW_ID		Account Id

Figure 99. Join Specification Definition

PAR_ROW_ID

The PAR_ROW_ID column identifies the primary key of the related record (Figure 100).

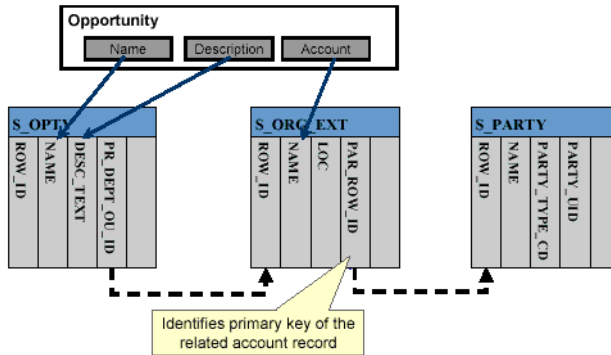


Figure 100. PAR_ROW_ID

Bringing Party Data into Party Business Components

For example, you can bring account data from the S_ORG_EXT table into the Contact business component to display in an applet, as shown in Figure 101.

Contacts							
Last Name	First Name	Middle Name	Mr/Ms	Email	Work Phone #	Job Title	Account
Aamot	Gina		Mr.	Gina_Aamot@aep.cc	(614) 343-8700	IT Manager	AEP Communication
Abanilla	David		Mr.	David_Abanilla@aep	(614) 343-8732	Proj Leader-A	AEP Communication
Abate	Laura		Ms.	Laura_Abate@aep.c	(614) 343-8723	Engineering M	AEP Communication
Abboline	Glen		Mr.	glen_abboline@siebe	(650) 295-5000	Global Service	eBusiness Beratur
Abdallah	Trey			Trey_Abdallah@aep	(770) 555-4565	Specifications	AEP Communication
Abel	Robert	P.	Mr.	Robert_Abel@aep.c	(860) 555-5186	Assistant Dire	AEP Communication
Abelman	Brent		Mr	abelman@atb.com	(650) 549-7311	Proj Mgr	Alberta Treasury Br

Figure 101. Bringing Party Data Into a Party Business Component

Join Definition

You should not use the implicit join for S_ORG_EXT that already exists. Both S_CONTACT and S_ORG_EXT are extension tables for S_PARTY. You must use an explicit join to S_ORG_EXT based on the appropriate foreign keys. It will have a different alias (Figure 102).

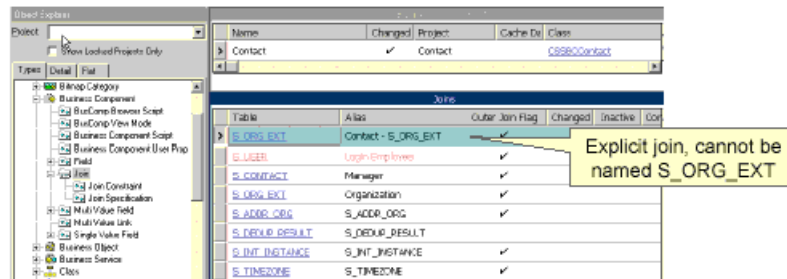


Figure 102. Join Definition

Mapping a Field to a Column in a Party Table

You must do the following:

- Create the required join if it does not exist.
- Create the single-value field.

Creating a Join to a Party Table

- Verify that the relationship is 1:1 or M:1.
- Identify the foreign key column for the desired relationship to the joined table.
- Create, if necessary, a field in the business component to reference the foreign key column.
- Create the join. Assign an appropriate alias property.
- Create the join specification. Use PAR_ROW_ID for the Destination Column.

Creating a Single-Value Field

- Select the appropriate explicit join (Figure 103).
- Select the desired column in the joined table.
- Set the appropriate type.

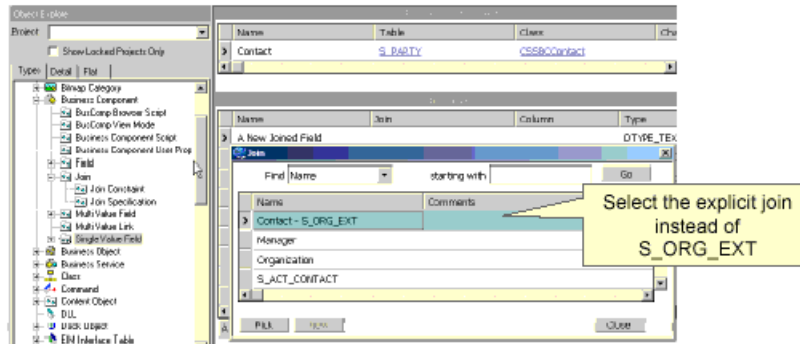


Figure 103. Mapping a Field to a Column in a Party Table

Links

A link implements a one-to-many (or master-detail) relationship between business components based on their base tables. The Link object type makes master-detail views possible, in which one record of the master business component displays with many detail business component records that correspond to the master. A master-detail relationship appears in [Figure 104](#), showing the Contact in the form and the many accounts assigned to contact in the list.

The form applet displays one record from the master business component.

The list applet displays all records from the detail business component that correspond to the master record.

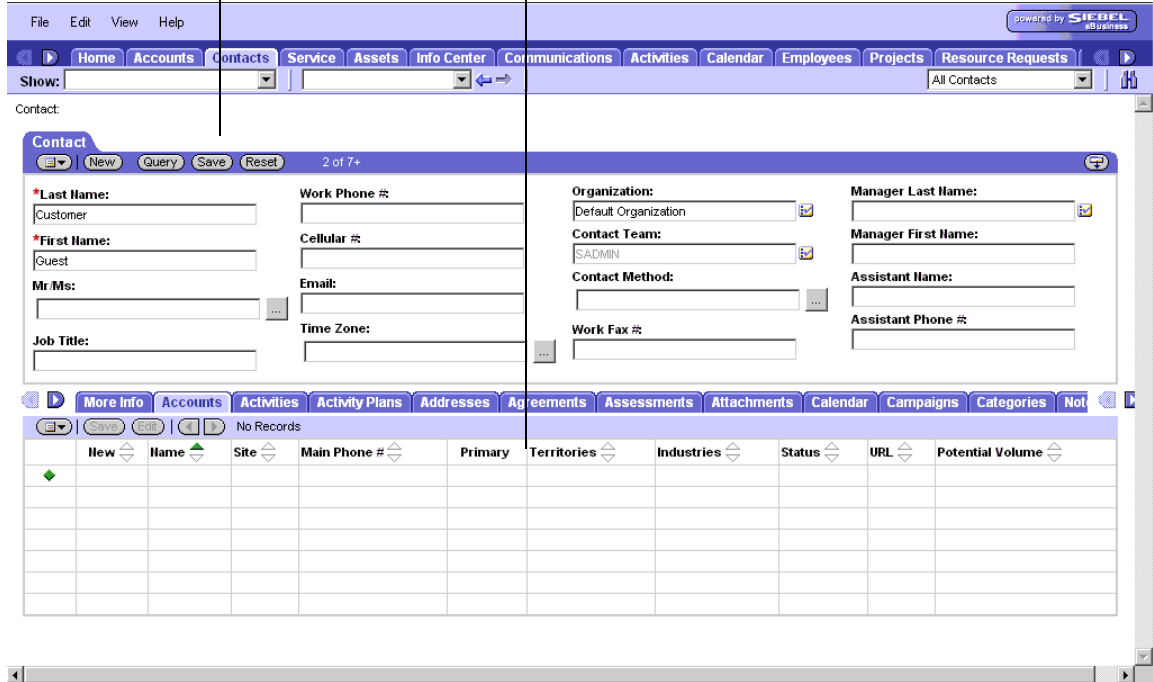


Figure 104. Link in a Master-Detail View

In this master-detail view, each Opportunity record can have many Product records. The synchronization between the master and detail business components in a master-detail view is accomplished with a link between the two business components and the inclusion of the link and business components in a business object. Business objects are described in [“About Business Objects” on page 418](#).

NOTE: Link destination fields are initialized automatically when you add a record to the child business component in a link.

Links are also used in the implementation of multi-value group applets. A multi-value group applet is a dialog box that displays multiple records of data associated with one control in the originating applet. For example, a multi-value group applet may be used to list the addresses, industries, or sales team members associated with an Account. A multi-value group applet for account addresses appears in [Figure 105](#).

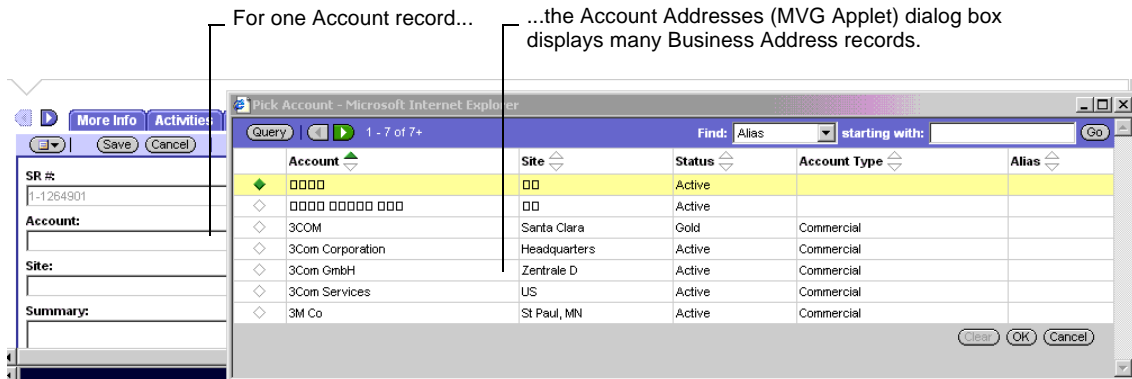


Figure 105. Multi-Value Group Applet Example

The relationship between the business component of the originating applet and the business component of the multi-value group applet is one-to-many; that is, a master-detail relationship. This master-detail relationship, as with all master-detail relationships in Siebel applications, is implemented through a Link object (in addition to other object types). A multi-value link is necessary to adapt a link for multi-value group applet use.

NOTE: The relationship between the two business components is one-to-many in the context of the multi-value link and multi-value group. There may be, in fact, a many-to-many relationship (for example, between opportunities and positions), but in the context of the multi-value group, only one master-detail relationship is presented.

How a Link Is Constructed

The relationships between object definitions used to implement a link appear in [Figure 106](#).

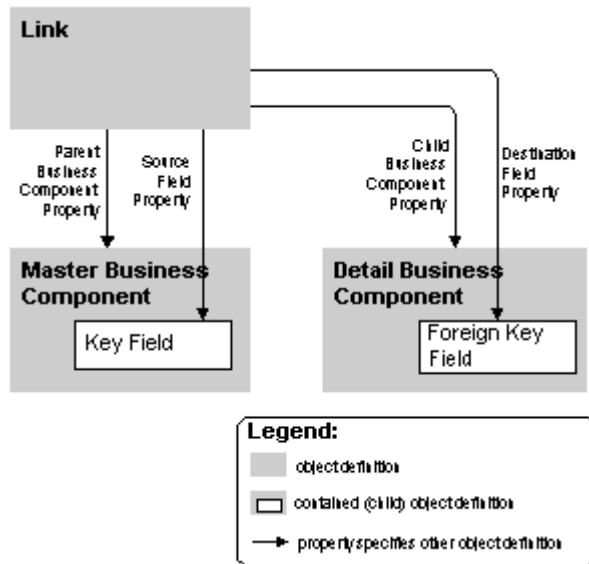


Figure 106. Link Property Relationships

The object definitions in [Figure 106](#) are as follows:

- **Link.** The Link object definition specifies a master-detail relationship between two business components. It identifies the master and detail business components, the key field in the master business component, and the foreign key field in the detail business component.
- **Master business component.** The master business component is the “one” in the one-to-many relationship. The name of this object definition is specified in the Parent Business Component property in the Link object definition.

- **Detail business component.** The detail business component is the “many” in the one-to-many relationship. The name of this object definition is specified in the Child Business Component property in the Link object definition.

NOTE: The Calendar business component should not be used as the master or detail business component in a link.

- **Source (primary key) field.** The source field, also known as the primary key field, is a field in the master business component that uniquely identifies records in the business component. It represents the ROW_ID column from the business component’s base table. The name of this field is specified in the Source Field property in the Link object definition. Source field typically, but not necessarily, represents the row id column from the business component’s base table.
- **Destination (foreign key) field.** The destination field, also known as the foreign key field, is a field in the detail business component that points back to the master record in the business component. Account Id and Opportunity Id are typical foreign key fields. A foreign key field represents a foreign key column from the detail business component’s base table, such as PR_DEPT_OU_ID (the base table for the Account business component). The name of this field is specified in the Destination Field property in the Link object definition.

In a link based on an intersection table, that is, one in which the Inter Table, Inter Parent Column, and Inter Child Column properties are non-blank, you do not specify the Source Field or Destination Field properties. Otherwise, the Destination Field property needs to contain the name of a field in the base table of the business component (not based on a join), and the field has to be updated.

NOTE: For a M:M link, you could specify a source field. Destination will always default to Id, even if another value is specified.

Using a Link in a Master-Detail View

A link in a master-detail view is implemented using the object types illustrated in Figure 107.

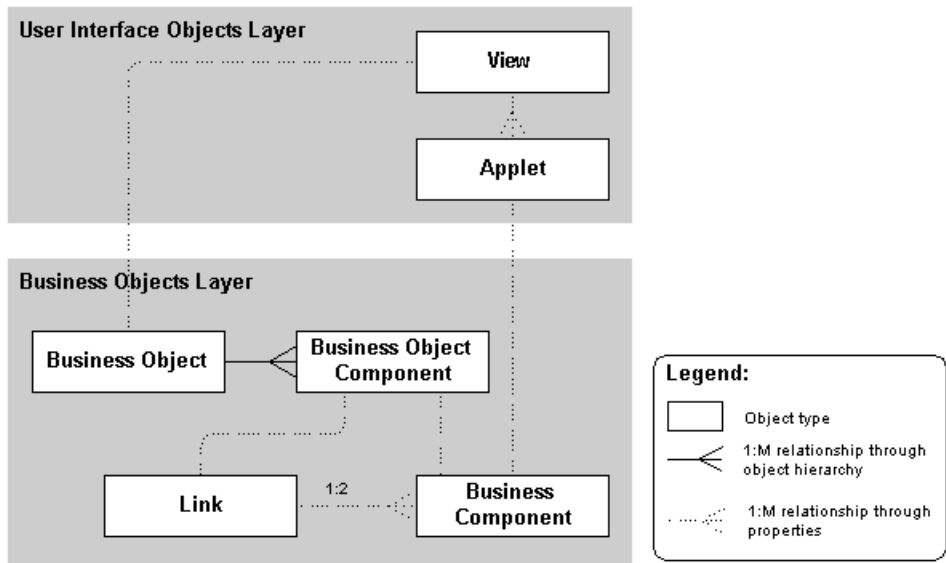


Figure 107. Link Architecture

In a master-detail view, a Link object definition is incorporated into a business object (by means of a Business Object Component object definition) to establish the master-detail relationship. This relationship applies to any use of the two business components together within the context of the business object. Each view specifies the business object it uses in its Business Object property. This forces the view to operate as a master-detail view, as specified in the link, without any additional configuration of the view. This is discussed in greater detail in “[Master-Detail Views](#)” on page 490.

Using a Link in a Multi-Value Group

A link object definition provides the basis for a multi-value link object definition, which in turn is used to implement a multi-value group applet. A multi-value group applet is a dialog box that provides the means to display and maintain multiple records of data associated with one control in the originating applet.

Multi-value links are described in [“Multi-Value Links” on page 400](#).

Using a Link in a Many-to-Many Relationship

Two link object definitions with opposite master-detail settings are used to establish a many-to-many relationship based on an intersection table. The Inter Table, Inter Parent Column, and Inter Child Column properties of the two Link object definitions are used to establish the connection between the links and the intersection table.

For information on this use of links and how to configure them for this purpose, refer to [“Intersection Tables” on page 204](#).

Using a Link When Merging Records

When you merge two records into a business component, any attached records (for example, the record for another business component) are not reassigned. For re-assignment to take place, you need to define a link between the two business components.

Cascade Delete Property

The Cascade Delete property in a Link object definition indicates what action to perform on detail business component records of the link if the master record is deleted. The following three values are available for this property:

- **CLEAR.** If CLEAR, the foreign key reference is removed if the master record is deleted, but the detail records remain in place.
- **DELETE.** If DELETE, the detail records are deleted along with the master.

- **NONE.** If NONE, no operations are performed on the detail record in response to deletion of the master, and the foreign key reference is not removed.

CAUTION: Do not use DELETE if the child business component in this link is also a detail business component in another link. In this case, you use CLEAR instead.

Cascade Delete is not available for many-to-many links. With a many-to-many link, Siebel applications will automatically delete the intersection record but will leave the child record intact, as it may have other parents.

When you delete a record which is pointed to by foreign keys of other tables, the references to it may or may not be deleted. If those references are not deleted, the user is left with row IDs which point to nonexistent records. In the case of multi-value groups, sometimes these foreign keys will be converted to say “No Match Row Id.”

Multi-Value Links

The Multi Value Link object type is a child object type of the Business Component object type. A *multi-value link* implements a special use of the Link object type, which is the maintenance by the user of a list of records attached to a control or list column in an applet. The group of attached detail records is called a *multi-value group*.

An example of a multi-value group is the Account Addresses dialog box invoked from the Address Line 1 text box in the Account Entry Applet. This is illustrated in Figure 108 and Figure 109.

The Address text box displays primary field value from the multi-value group.

The Account Addresses dialog box invokes the MVG applet and displays the primary Address value.

Primary	Address Line 1	Address Line 2	City	State	Zip	Country	Global Owner
✓	5445 Dtc Pkwy Ste		Englewood	CO	80111-3056	USA	
	1787 Sentry Pkwy		Blue Bell	PA	19422-2240	USA	
	10 Riverview Dr		Danbury	CT	06810-6268	USA	
	570 Lake Cook Rd		Deerfield	IL	60015-5612	USA	
	33 W Monroe St		Chicago	IL	60603-5300	USA	
	44 Old Ridgebury Rd		Danbury	CT	06810-5107	USA	
	33 W Monroe		Lake Forest	IL	60045	USA	

Address Line 1: 5445 Dtc Pkwy Ste 1100
 Address Line 2:
 City: Englewood
 State: CO

Figure 108. Multi-Value Group Example

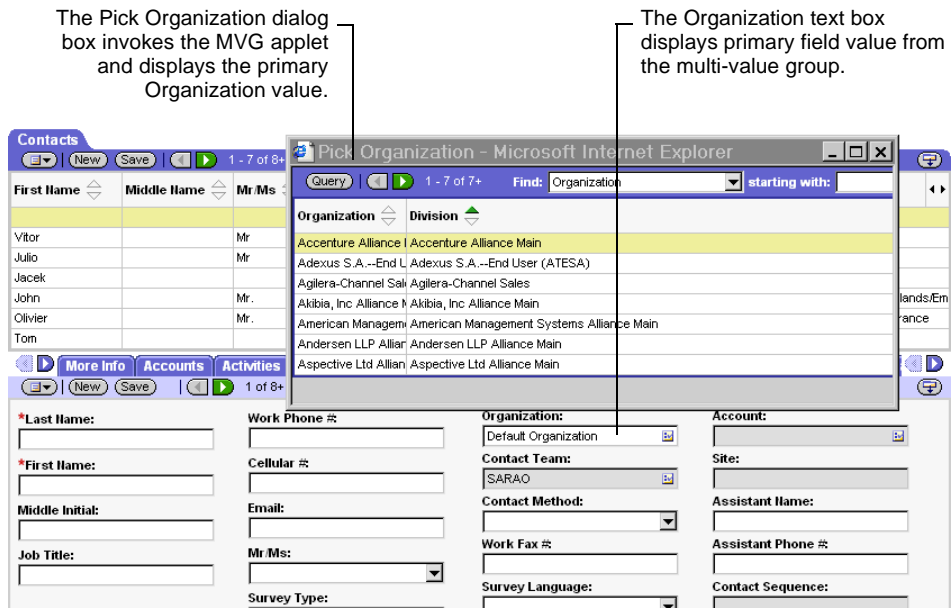


Figure 109. Multi-Value Group Example

An account can have multiple addresses. These are stored in the Business Address business component. Clicking the Select button to the right of the Address text box brings up a dialog box. This dialog box lists the attached addresses, including the street address, city, state, and ZIP Code for each. It also provides the means for the user to add, delete or modify individual records.

In the Account Form Applet, the Address, City, State, Zip and Country text boxes display the values from the corresponding fields in the primary record in the Business Address business component. The primary record is indicated in the multi-value group applet with a checkmark in the list column labeled Primary. The user can select a different primary record by clicking the Primary list column in a different record.

The fields in the master business component (Account in the illustration) that are populated by the primary record in the multi-value group business component are called *multi-value fields*.

NOTE: If you want to query the originating applet for all master records that have a detail record with a specific field value you can only use Multi-value fields.

Multi-value fields are populated with data from a record in the detail business component because of the multi-value link. Multi Value Link is a child object type of Business Component that defines a master-detail relationship (based on a link) to embed in the business component. These embedded master-detail relationships are used to expose fields from the detail business component as fields directly in the master business component.

NOTE: Most, but not all, multi-value links are set up to designate a primary record. Those that do not designate a primary use the first record retrieved from the detail business component. For more information, refer to [“Primary ID Field” on page 413](#).

How a Multi-Value Link Is Constructed

A multi-value link is based on a link object definition; the link is referenced in the Destination Link property of the Multi Value Link object definition. It is the link object definition that specifies the one-to-many relationship between the master and detail business components. The multi-value link object definition performs two roles:

- To give fields in the master business component access to primary record field values through the link.
- To allow embedding of detail data in the same business component as master data, so both can appear in the same applet.

The object types illustrated in [Figure 110](#) participate in the configuration of a multi-value link.

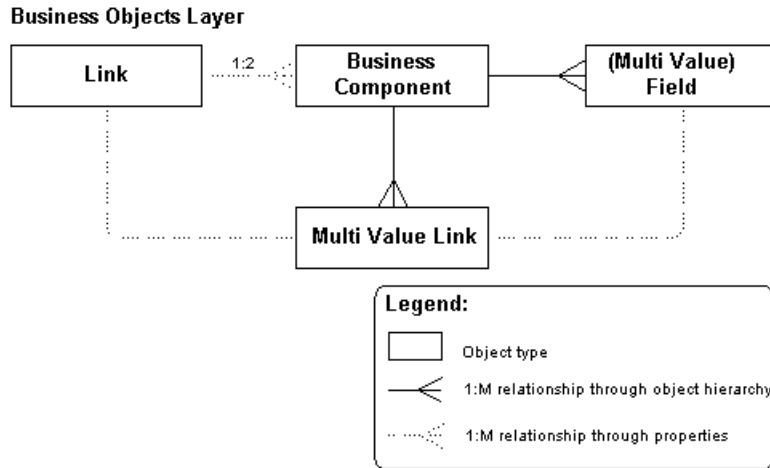


Figure 110. Multi-Value Link Architecture

The object type box in the diagram labeled (Multi Value) Field indicates that either field or multi-value field is correct for referring to this object type in this context. Multi Value Field is a distinct object type, but only in the sense that it can be accessed in the Object Explorer. It is only a representation of the Field object type. Multi-value fields are those fields that have a non-blank Multi Value Link property and a Multi Valued property value of TRUE; all other fields are single-value fields.

The details of the object definition relationships appear in [Figure 111](#).

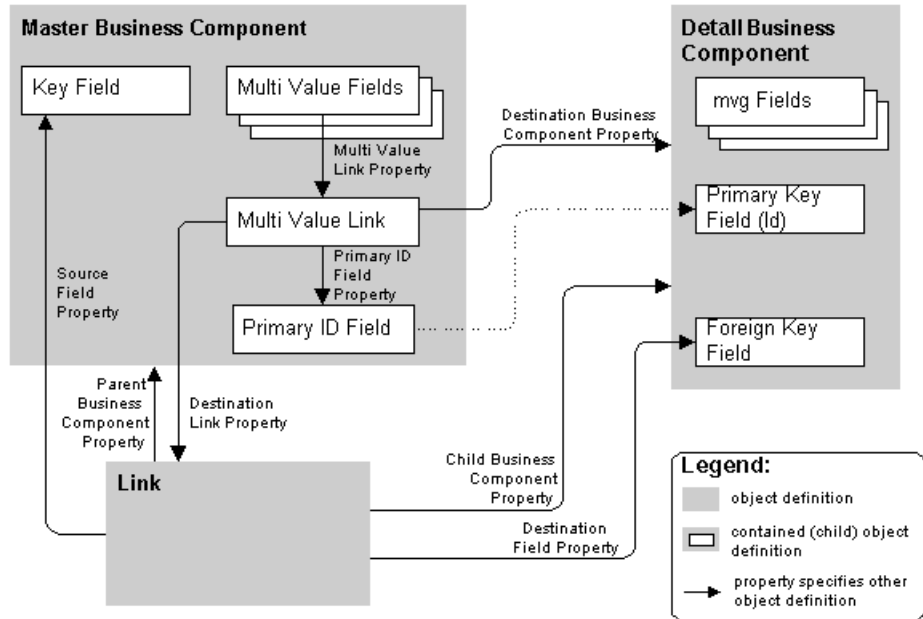


Figure 111. Multi-Value Link Details

The roles of the object definitions in [Figure 111](#) are explained in the following list. References to the address example refer to the Account Addresses dialog box illustrated in [Figure 108 on page 401](#).

- Master business component.** The master business component is the “master” in the master-detail relationship specified in the link. Fields from this business component are displayed in the applet from which the multi-value group applet is initiated. The master business component in the Account Addresses dialog box example is Account.

NOTE: The Account Addresses dialog box example is illustrated in [Figure 108 on page 401](#). The explanations of object definitions below refer to this example.

- **Multi-value fields.** Multi-value fields are fields in the master business component that are populated by the current (typically primary) record in the detail business component through the multi-value link and link object definitions. Each of these fields has the name of the multi-value link specified in its Multi Value Link property, and a Multi Valued property setting of TRUE. A multi-value field has a blank Column property setting because its values are obtained from the current record in the detail business component, rather than from the master business component's base table.
- **Key field.** The key field in the master business component is the primary key for that business component. The key field is referenced in the Source Field property of the Link object definition.
- **Multi Value Link object.** The Multi Value Link object definition defines the relationship between the link object definition and fields in the master business component, using the following properties:
 - **Destination Link.** Identifies the link.
 - **Destination Business Component.** Identifies the detail business component.
 - **Primary Id Field.** Identifies the field in the detail business component that designates which record is the primary.

In the example, the multi-value link is called Business Address.

- **Link.** The Link object definition specifies a master-detail relationship between the two business components. The Link object definition can be used in other contexts, such as master-detail views or other multi-value links. The multi-value link identifies the link in its Destination Link property. In the address example, the link is Account and Business Address.
- **Detail business component.** The detail business component supplies the detail records in the master-detail relationship. In the address example, this is Business Address.

- **Foreign key field.** The foreign key field contains row ID values that point back to records in the master business component and uniquely identify the master for each detail business component record. The foreign key field is used in the specification of the link; the link identifies the foreign key field in its Destination Field property. In the address example, the foreign key field is Account Id.

NOTE: There is no foreign key field specified in a link based on an intersection table.

- **Primary ID field.** The primary ID field in the master business component holds the row ID value of the primary record for each multi-value group in the detail business component. It is identified in the Primary Id Field property of the multi-value link. The primary ID field allows the primary detail record to be identified for each master record. For more information, refer to [“Primary ID Field” on page 413](#).

How an Indirect Multi-Value Link Is Constructed

If there is a join object definition that joins a master business component to the master business component of the link, the existing link can be used in a multi-value link. In essence, the multi-value link is being based on a “join to a link.” Such a multi-value link is known as an *indirect multi-value link*. The configuration of an indirect multi-value link is illustrated in [Figure 112](#).

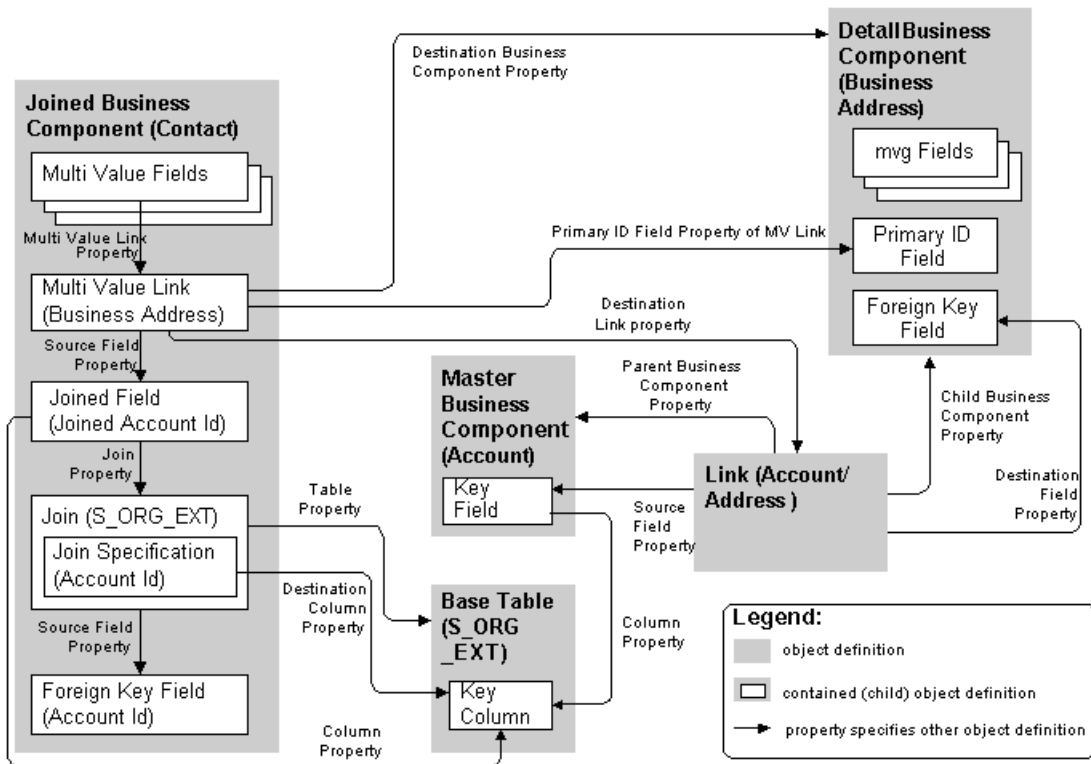


Figure 112. Indirect Multi-Value Link Details

In [Figure 112 on page 408](#), the object definition names have been provided for an example multi-value link called Business Address in the Contact business component. Although given the same name as its counterpart in the Account business component, this is a different object definition. For a comparison of conventional and indirect multi-value links, review [Figure 111 on page 405](#).

The primary difference between the Business Address multi-value link in the Contact business component and in its Account counterpart is that the multi-value link object definition is found in a business component other than the master business component.

The Source Field property in the multi-value link in the Contact business component is non-blank. In a conventional multi-value link this property is blank, indicating that the Id field in the current business component is used (corresponding to ROW_ID in the base table). In the indirect multi-value link, the Source Field property specifies a field in the S_ORG_EXT join called Joined Account ID. The Joined Account ID field provides the Account Id of the Account that corresponds to the current Contact.

The roles of the object definitions in [Figure 112 on page 408](#) are explained as follows:

- **Join business component.** The join business component has a master-detail relationship with the master business component in the link. In this relationship, the join business component is the detail rather than master. The indirect multi-value link is established as a child object definition of the join business component.
- **Multi-value fields.** Multi-value fields are fields in the join business component that are populated by the primary record in the detail business component through the multi-value link and link object definitions. Each field has the name of the multi-value link specified in its Multi Value Link property, a Multi-Valued property setting of TRUE, and a blank Column property.
- **Multi-value link.** The Multi-Value Link object definition defines the relationship between the link object definition and fields in the master business component, using the following properties:
 - **Destination Link.** Identifies the link.
 - **Destination Business Component.** Identifies the detail business component.
 - **Primary Id Field.** Identifies the field from the business component that the MVL belongs to.

- **Joined field.** In a conventional multi-value link, the Source Field property is blank. In an indirect multi-value link, the Source Field property specifies a joined field in the same business component as the multi-value link. The joined field represents the ROW_ID column from the base table of the master business component. The ROW_ID column is obtained by means of a join.
- **Join and join specification.** The Join and Join Specification object definitions make it possible to populate the joined field.
- **Foreign key field (in the joined business component).** The foreign key field represents a foreign key column in the base table. The foreign key field points to rows in the joined table, in this case the base table of the master business component. The foreign key field is used in the implementation of the join.
- **Master business component.** The master business component is the “master” in the master-detail relationship specified in the link. The master business component in the example in [Figure 112 on page 408](#) is Account.
- **Base table.** The join, join specification, and foreign key field in the join business component access the base table of the master business component. This makes possible a join relationship that provides a master business component record and, indirectly, a set of detail business component records for each join business component record.
- **Key field.** The key field in the master business component is the primary key for that business component. The key field is referenced in the Source Field property of the Link object definition.
- **Link.** The Link object definition specifies a master-detail relationship between the master and detail business components.
- **Detail business component.** The detail business component supplies the detail records in the master-detail relationship.
- **Foreign key field (in the detail business component).** The foreign key field contains row ID values that point back to records in the master business component. These row ID values uniquely identify the master for each detail business component record. The foreign key field is identified in the link in the Destination Field property.

- **Primary ID field.** The primary ID field in the master business component holds the row ID value of the primary record for each multi-value group in the detail business component. The primary ID field is identified in the Primary Id Field property of the multi-value link. The primary ID field allows the primary detail record to be identified for each master record. For more information, refer to [“Primary ID Field” on page 413](#).

The parent component of a multi-value link is usually the same as the business component in which the MVL is defined. However, by using the Source Field property of the Link object, it is also possible to create an MVL whose parent business component is related to the current business component indirectly using a join or another MVL.

Every MVL in a Siebel application is based on an underlying Link object, whose name is specified by the Destination Link property of the multi-value link. Every link, in turn, defines a one-to-many relationship between two business components. Typically, the business component in which an MVL is defined is the same as the parent business component of the underlying link on which the MVL is based.

For example, consider the Business Address multi-value link in the Account business component:

```
[MultiValueLink]

DestBusComp = "Business Address"
DestLink = "Account/Business Address"
PrimaryIdField = "Primary Address Id"
CheckNoMatch = "TRUE"
PopupUpdOnly = "TRUE"
NoCopy = "TRUE"
```

The Destination Link property indicates that this MVL is based on the Account/Business Address link, which is itself defined as:

```
[Link]

Name = "Account/Business Address"
ParentBusComp = "Account"
ChildBusComp = "Business Address"
DestField = "Account Id"
CascadeDelete = "Delete"
```

The parent business component of this link is the Account business component, which is also the business component in which the MVL has been defined. In this typical MVL configuration, the multi-value group will be populated with all the children Business Address records for whichever Account is currently selected in the Account business component.

Indirect Multi-Value Links Using Joins

Although the parent business component of a multi-value group is usually the same as the business component in which the multi-value link 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, however, the Business Addresses are not directly related to the Opportunities themselves; instead, they are children records of whatever Account is associated with the current Opportunity (if there is such an Account). In order for Siebel applications to populate this MVG correctly, it needs to know how to find the appropriate parent Account record for the link given the current record in the Opportunity business component. The Source Field property of the Link object exists for this purpose.

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

```
[MultiValueLink]

SrcField = "Joined Account Id"
DestBusComp = "Business Address"
DestLink = "Account/Business Address"
PrimaryIdField = "Primary Address Id"
CheckNoMatch = "TRUE"
PopupUpdOnly = "TRUE"
```

The Destination Link property of this MVL is still the Account and Business Address link, which defines the one-to-many relationship between Accounts and Business Addresses. Therefore, 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).

In order to determine the appropriate Account record for which to get the children Business Addresses, the Siebel application looks at the Source Field property of the MVL. For this particular multi-value link, the Source Field 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. Therefore, for each Opportunity record, the Siebel application 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.

Primary ID Field

The Link and Multi Value Link object definitions have a set of properties that you can use to specify to the system how to obtain the record ID of the first record to display of the detail table each time the master record changes. These properties are Primary Id Field, Use Primary Join, and Auto Primary. Together they implement the primary ID field.

The basic concept behind a primary ID is that it is faster for a Siebel application to retrieve one primary record from the MVG business component through a join than retrieve all of them through a sub-query—especially since users can see values from only one child record until they open up the MVG applet. To create a primary field for a one-to-many or many-to-many relationship, complete the following procedure.

To configure a primary field for a 1:M or M:M relationship

- 1** Create a Primary Id column.
- 2** Create a field based on that Primary Id column.
- 3** In a Multi-Value Link, set the Primary Id Field attribute to the new Primary Id field.
- 4** Set the Use Primary Join attribute to TRUE.

For example, in the Account business component the primary ID field for the Address multi-value group is called Primary Address Id. The Account Address Mvg Applet displays the corresponding multi-value group. The primary record, indicated with a checkmark in the list column labeled Primary, has its row ID stored in the Primary Address Id field in the account record. Each time there is a different account record displayed, the multi-value fields for the Address load the primary Business Address record's values only. It is not necessary to query the Business Address business component for multiple rows. This can be a significant performance enhancement, especially in list applets.

NOTE: In a multi-value group applet, the list column that displays the check mark (indicating the primary or nonprimary status of each record) obtains its data from a system field called SSA Primary Field. This field does not appear in the Object Explorer or Object List Editor, but may be referenced by a list column for this purpose.

The benefit of using a primary ID, from the system's standpoint, is that it converts a one-to-many relationship into a one-to-one relationship. This allows the row retrieval process to be simplified from a query with subqueries to a simple join query. This substantially improves performance, especially when the user is scrolling through the records of a list applet that displays the master.

The properties of Link or Multi-Value Link object types used to implement a primary ID field are as follows:

- **Primary ID Field.** This property specifies the name of the field in the master business component that holds the row ID values pointing to primary records in the detail business component.
- **Use Primary Join.** The Use Primary Join property is a TRUE or FALSE property that turns the Primary Join feature on or off. If TRUE, the primary detail record is obtained for each master record through a join on the primary ID field. If FALSE, the detail table is queried again with each master record change.

- **Auto Primary.** This property setting determines how row ID values are populated in the primary ID field, based on a system-supplied list column labeled Primary in the multi-value group applet. The user can manually select the primary. Auto Primary determines how, if at all, the primary selection is defaulted. The possible values for Auto Primary are DEFAULT, SELECTED, or NONE as follows:
 - **DEFAULT.** The first record automatically becomes the primary.
 - **SELECTED.** The highlighted record becomes the primary when the user views the multi-value group applet and then exits.
 - **NONE.** The user must manually specify the primary.

SELECTED only pertains when there are several multi-value links pointing to the same detail business component. This is the case for the Bill To Business Address and Ship To Business Address multi-value links in a standard Siebel Sales application. These multi-value links exist under both the Order and Account business components. In this case, an example of the desired behavior is as follows: if a primary is not set for the Bill To address, then when the Siebel application does a separate query to bring back all addresses associated with the account (or order), it will check to see whether one of the addresses has already been selected as primary for the Ship To address and, if so, it will SELECT (that is, set) that address as the primary for Bill To address as well.

When the Auto Primary property of a Multi Value Link object has a value of SELECTED, setting read-only properties at the applet level still does not force the SSA Primary Field to be read-only.

NOTE: If the destination business component of the Multi-Value Link is read-only, you may receive the following error message, “This operation is not available for a read-only field ‘SSA Primary Field’.” This is because the Primary ID field is automatically updated through the system field ‘SSA Primary Field’, which belongs to the destination business component. Additionally, if this business component is read-only, the field is read-only as well and cannot be updated.

Allowing Users to Set Primaries

You can set the MVG Set Primary Restricted: *visibility_mvlink_name* user property in the business component underlying the MVG applet to allow certain users to set primaries. Setting this user property to FALSE allows the Primary team member to be altered by someone other than the Manager or Siebel Administrator.

If this user property is not set, only Siebel Administrators (in Admin mode) and Managers (in Manager view mode) have the ability to change the Primary team member on opportunities, accounts and contacts.

For more information, see *Configuration Guidelines*.

Using the Check No Match Property with a Primary Join

When a multi-value link has been configured with a primary join—which is the typical situation—there are circumstances in which the foreign key used by this join to identify the primary record is unable to find the primary. For example, this can happen when the primary record has been deleted from the multi-value group or the multi-value group is newly created and has no records. In such cases, the multi-value link can be configured to update the primary foreign key to a value of NULL, or to a special value of NoMatchRowId, depending on your requirements. This behavior is configured through the Check No Match property of the Multi Value Link object type, and has performance consequences.

The purpose of the special “NoMatchRowId” value is to prevent secondary queries on foreign key values that are known to have failed, thereby improving performance, much in the same way that using a primary join improves performance.

The NoMatchRowId generating and testing behavior is activated by setting Check No Match to FALSE for the MVL. This setting has the following results:

- When the application encounters a master record where the primary foreign key is NULL or invalid, it performs a secondary query to determine if there are detail records in the multi-value group. If it finds there are no detail records, it sets the primary ID field to the special value NoMatchRowId.
- When the application encounters a master record where the primary foreign key has the value “NoMatchRowId,” this indicates to the system that there are no detail records in the multi-value group and the secondary query is not performed.

If you set Check No Match to TRUE, the Siebel application will perform a secondary query whenever the outer join on the primary fails, or is set to NULL or NoMatchRowId. If the secondary query finds a matching detail record, it updates the foreign key with that record's row ID, provided the MVL has an Auto Primary property setting of DEFAULT. If no matching child record is found, or Auto Primary is set to NONE, the application leaves the existing value intact.

A Check No Match setting of TRUE can have serious negative performance consequences. If a multi-value group is sparsely populated (that is, most master records do not have any detail records in the multi-value group) and has Check No Match set to TRUE, it will be almost as slow as not having a primary join at all.

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

The Use Primary Join property should be set to TRUE if CheckNoMatch is TRUE. If CheckNoMatch is set to TRUE and Use Primary Join is FALSE, then the Siebel application will always do the secondary query to find the child records.

How a Cascade Copy with a Multi-Value Link Is Constructed

It is often desirable to be able to configure a business component to support the copying of its detail records when one of its records is copied. You implement this with a feature called *cascade copy*. For example, when you make a copy of an opportunity record to create a similar opportunity, you may always want the list of contacts for that opportunity copied with it.

To implement the cascade copy feature for a business component, you create a Multi Value Link child object definition and specify the following properties:

- **Destination Link.** The name of the link in which the master-detail relationship is specified.
- **Destination Business Component.** The name of the detail business component.

- **No Copy.** This must be set to FALSE and #Field ID should be set to No Copy = FALSE. If the No Copy property is set to TRUE, cascade copying is disabled. However, an exception to this occurs when the corresponding field is defined as the Destination field in a link. In this case, the link automatically populates the field and ignores the value of the No Copy property.

A multi-value link used in the implementation of a multi-value field automatically copies the detail records (unless disabled with No Copy) because it is assumed that a multi-value group travels with its parent record. For example, you would normally want the account addresses, sales team, and industry list for an account to copy with that account.

This capability is used for a different purpose when cascade copy is implemented for a multi-value link not used in a multi-value field. The multi-value link does not need to be attached to a field in the business component, or used in a multi-value group. It just needs to be created as a child object definition of the master business component, configured to point to the detail business component and link, and set with copying enabled (a No Copy value of FALSE).

Cascade copy can be implemented for a many-to-many relationship, that is, where the destination link has a non-blank Inter Table property value. In this circumstance, new intersection table rows are created rather than new detail business component records. New associations are created rather than new records. These associations are between the new master and the existing detail records.

NOTE: Cascade copy has the potential to violate the requirement of uniqueness of values in indexes. For this reason, if copying the detail records would cause any unique index violations, the copy operation is aborted by the system.

About Business Objects

A business object implements a business model (as represented in a logical database diagram), tying together a set of interrelated business components using links. The links provide the one-to-many relationships that govern how the business components interrelate in the context of this business object.

The set of relationships established in a business object provides the foundation for views and screens. For example, [Figure 113](#) shows the Contact Detail - Opportunities View, a view that operates based on a one-to-many relationship defined in the Contact business object.

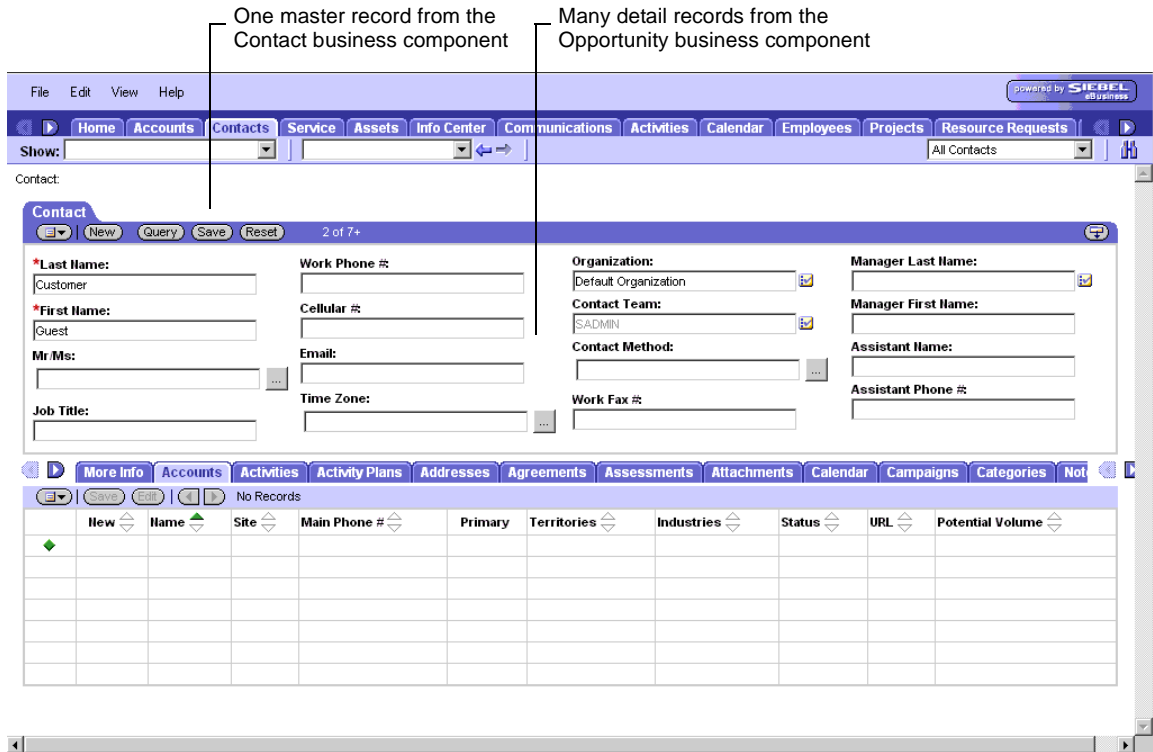


Figure 113. Master-Detail View Based on a Business Object

Every view has a business object assigned to it. A master-detail view can implement only a one-to-many relationship supported by its underlying business object. For example, the view in [Figure 113](#) can display a one contact to many opportunities relationship because Contact and Opportunity have this kind of relationship in the Contact business object, and the view (Contact Detail - Opportunities View) uses the Contact business object. In order to implement a view displaying the reverse relationship (one Opportunity master record to many Contact detail records), the Opportunity (rather than Contact) business object would be required as the business object of the view.

[Figure 114](#) displays the abstract relationships between the Business Object object type and two user interface object types, View and Screen.

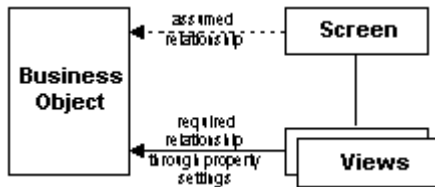


Figure 114. Relationship Between Business Object, Screen, and Views

Many views are built based on the same business object. Typically only one screen is associated with one business object. A business object is not assigned to a screen through a property setting the way a business object is assigned to a view. The relationship between a business object and a screen is an informal one dictated by good design practice, and it is not strictly enforced by the Siebel Tools software. In general, all of the views in a screen are implementations of the same business object.

The logical database diagram in [Figure 115 on page 421](#) illustrates the complete set of one-to-many relationships between business components in a single business object. In this case, it is the Account business object.

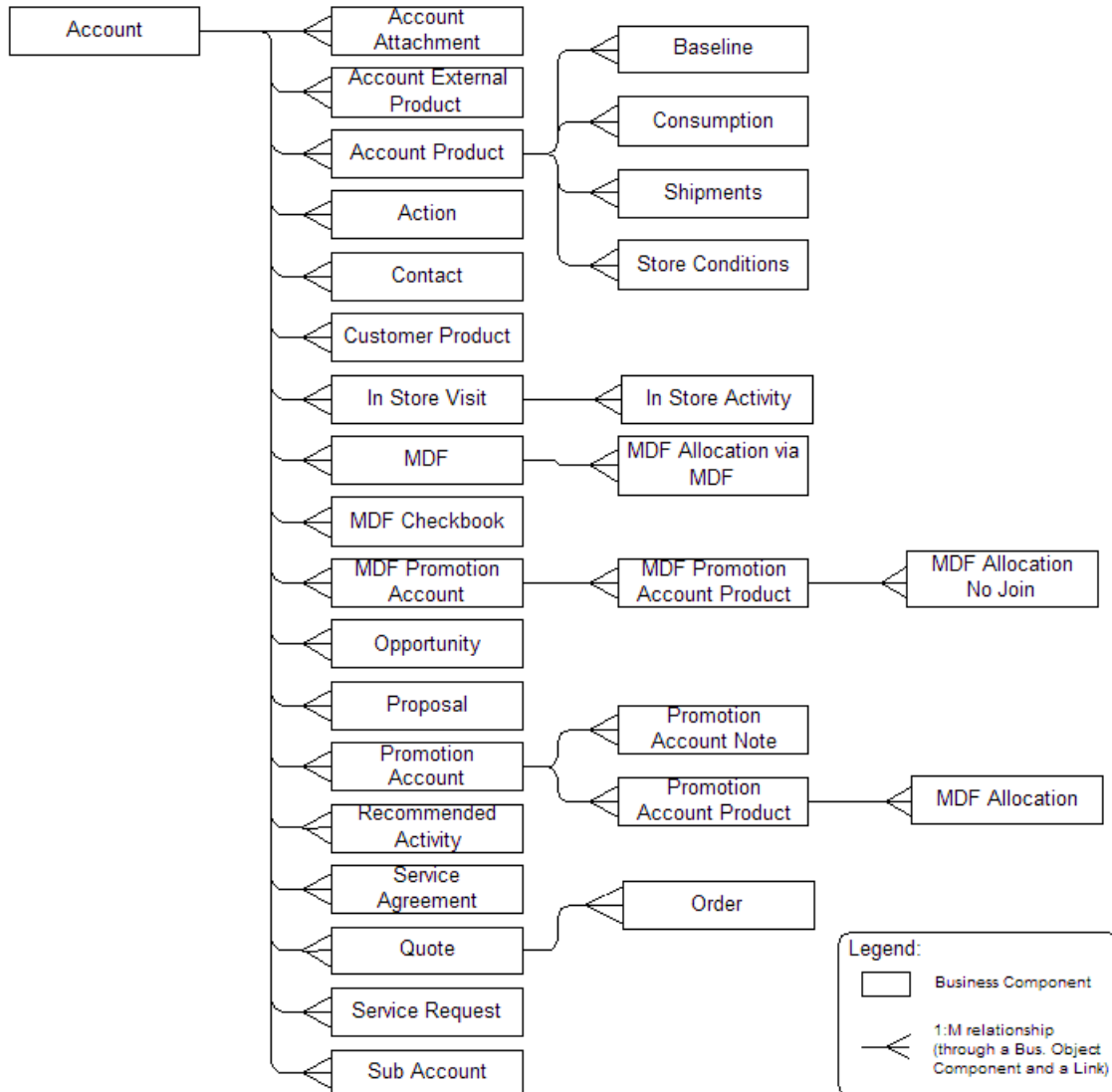


Figure 115. Master-Detail Relationships in the Account Business Object

The boxes represent business components, and the three-way connectors represent one-to-many relationships. A one-to-many relationship is implemented with a business object component and a link, as is explained in [“How a Business Object Is Constructed” on page 423](#).

The business object collects a logical grouping of business components and a set of links that associate them. Some of the same business components and links may appear in other business objects. The same two business components may have a one-to-many relationship in one business object, and the opposite one-to-many relationship (or no relationship) in another business object.

However, within the context of one business object, there is an unambiguous set of relationships between the business components in the grouping. When a particular business object is active because a view that uses it is active, the population of data records in business components in the business object is based on the relationships in the business object.

The benefit of business objects is reusability. The same business component can be used in various different sets of relationships by including it in multiple business objects.

NOTE: Not all business components included in a business object participate in master-detail relationships. Business components that are not part of the business model may also be incorporated in the business object. A Business Component object makes such a business component available for use in views based on the specified business object.

How a Business Object Is Constructed

The object types illustrated in [Figure 116](#) participate in the configuration of a business object.

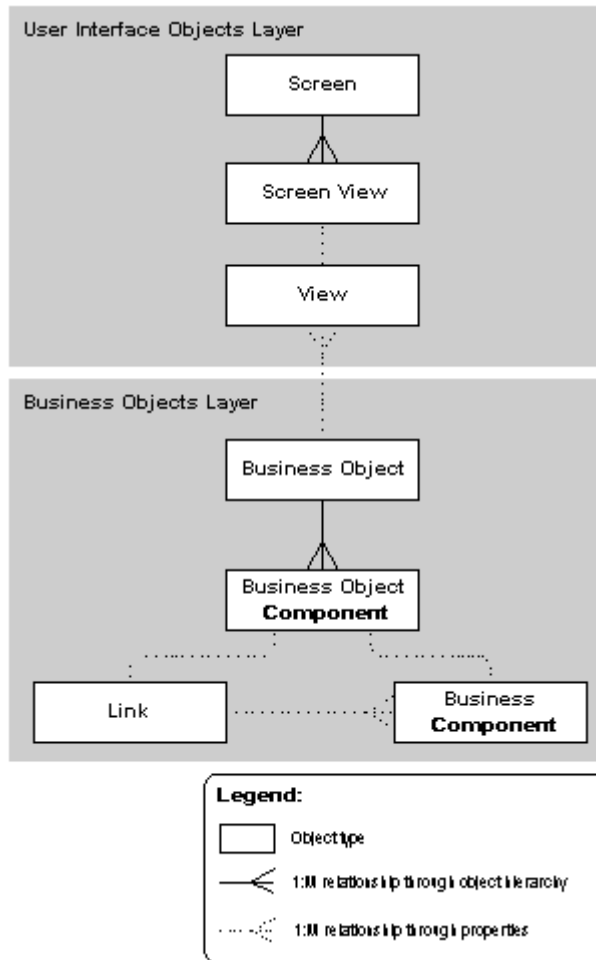


Figure 116. Business Object Architecture

The relationships between object definitions used to implement a business object appear in [Figure 117](#).

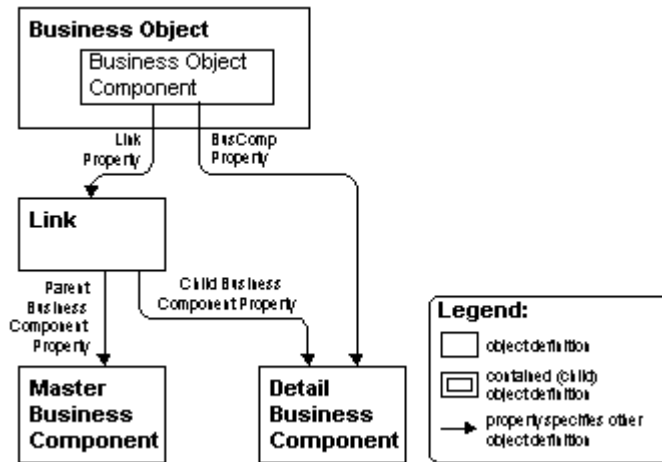


Figure 117. Business Object Details

The object definitions in [Figure 117](#) are described as follows:

- **Business Object object type.** The business object is a parent for multiple business object component child object definitions. Each business object component specifies a master-detail relationship. View object definitions reference the business object in their Business Object property.
- **Business Object Component object type.** Business Object Component is a child object type of Business Object. Typically, each business object component defines one master-detail relationship within the parent business object. Two properties within the business object component specify this relationship:
 - **Link.** Identifies the link object definition.

- **BusComp.** Identifies the detail business component object definition.

A business object component can be used to include a business component in the business object without a link. To accomplish this, you enter a blank value in the Link property of the business object component. A link-free business object component allows you to incorporate a business component in the business object for use in views based on the business object, even though the business component does not have one-to-many relationships with other business components in the context of that business object.

- **Link object type.** One link is referenced by each business object component. The Link object definition specifies the master-detail relationship that is being included in the business object by way of the business object component. Links are described in [“Links” on page 393](#).
- **Master business component.** The master business component is the “one” in the one-to-many relationship specified in the link. The Parent Business Component property in the Link object definition specifies the master business component.
- **Detail business component.** The detail business component is the “many” in the one-to-many relationship. The detail business component is specified both in the Child Business Component property of the Link object type and in the BusComp property of the Business Object Component object type.

Business Services

A business service is an object that encapsulates and simplifies the use of some set of functionality. Business components and business objects are objects that are typically tied to specific data and tables in the Siebel data model. Business services, on the other hand, are not tied to specific objects, but rather operate or act upon objects to achieve a particular goal.

Business services can simplify the task of moving data and converting data formats between the Siebel eBusiness Application and external applications. Business services can also be used outside the context of Siebel EAI to accomplish other types of tasks, such as performing a standard tax calculation, shipping rate calculation, or other specialized functions.

You can create business services in the Siebel client or in Siebel Tools. These services can then be accessed by Siebel VB or Siebel eScript code that you write and call from workflow processes. For the purposes of your integration projects using Siebel EAI, you must use Siebel eScript to write your scripts. Siebel EAI uses business services within a workflow pipeline.

For information about business services in general, see *Integration Platform Technologies: Siebel eBusiness Application Integration Volume II*. For information on creating workflow processes, which use business services you create, see *Siebel Business Process Designer Administration Guide*. See *Siebel VB Language Reference* and *Siebel eScript Language Reference* for more information on accessing property sets and business services with Siebel VB and Siebel eScript.

This chapter gives you information on configuring the business objects layer of the Siebel Web application. You should be aware that configuration done at the business objects layer might need to be exposed through the UI objects. The methods used to determine whether this is necessary, as well as additional configuration guidelines, are covered in this chapter and in subsequent chapters.

About the Application Development Process

The general sequence of application development tasks starts from the bottom layers and proceeds to the layers above. For example, a generalized sequence of tasks might look like the following:

- 1** Create or modify objects in the Business Object Layer:
 - a** Business components
 - b** Links
 - c** Business objects
- 2** Create or modify objects in the User Interface Layer:
 - a** Applets
 - b** Views
 - c** Screens
 - d** Applications

You must define business components, links, applets, and views in the order shown, or you will encounter application errors. It is also recommended that you define business objects, screens, and applications in the order shown, and then refine their definitions, as required, during the application development cycle.

This chapter covers configuration work at the business objects layer. Subsequent chapters discuss configuring objects in the user interface layer.

Defining Business Objects

The Business Objects Layer of the Siebel application architecture is where your application's data organization is defined.

You need to develop object definitions in this layer before developing definitions in the User Interface Layer.

The following Business Object Layer object types (introduced in [Chapter 2, “Siebel Architecture \(Basic Concepts\)”](#)) are the foundation of all Siebel applications:

- Business components
- Fields (child objects of business components)
- Business objects

Business component and field definitions in the Business Objects Layer map to table and column definitions in the Data Objects Layer.

Figure 118 summarizes the mappings required between user interface objects and business objects.

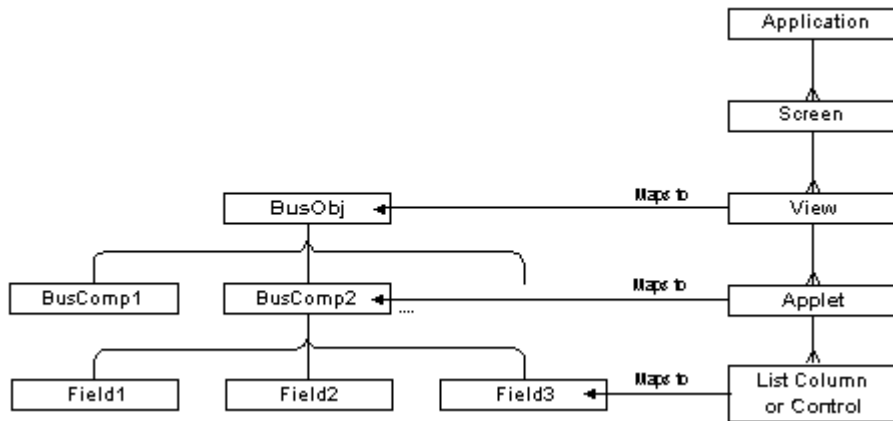


Figure 118. Mapping of UI Layer Objects to BO Layer Objects

To do the mapping you need to have knowledge of the underlying data model.

Usage and Configuration of Non-Licensed Objects

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

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

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

Development Sequence for Defining Business Objects

NOTE: Follow this sequence when you work with objects in the Business Objects Layer (the assumption is that you have already created a project and checked it out, as described in [Chapter 18, “Application Development Projects”](#)):

- 1** Create a business component (or modify an existing one) and add it to the project.
- 2** Add fields to the business component.
- 3** Create links that will relate the business components to one another within a business object.
- 4** Create a business object and add it to the project.
- 5** Associate the business component with the business object.
- 6** Validate your business object definitions.

These steps are discussed in more detail in the following sections.

Creating or Modifying a Business Component Definition

General instructions for modifying or creating object definitions are provided in “Modifying, Copying, and Creating New Object Definitions” on page 117.

Figure 119 shows the object definition for a business component called Account Category.

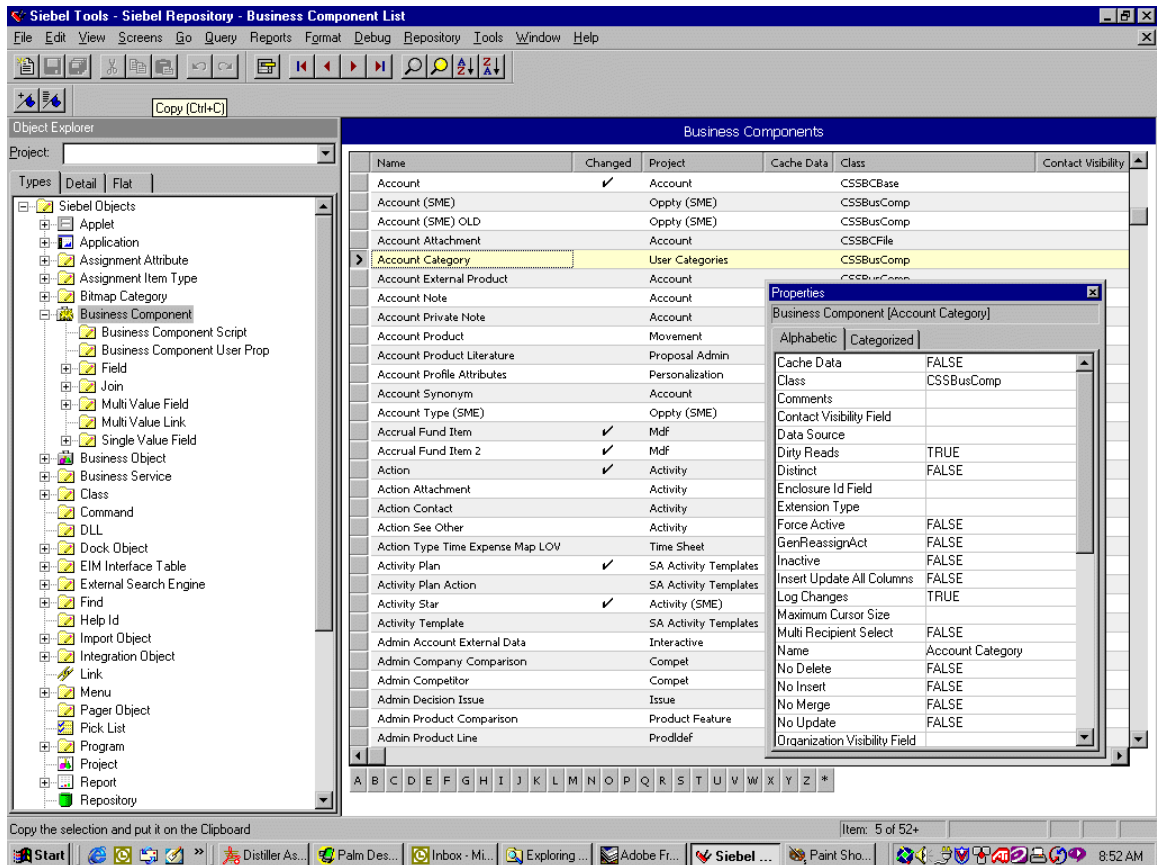


Figure 119. Business Component Object Definition: Account Category

Business Component Properties

Following are descriptions for key properties for the Business Component object type. For a complete list and more detailed descriptions, see *Object Types Reference*.

- **Class.** The C++ class that implements the functionality of the business component.

A picklist of values appears for setting this property.

Siebel applications have a hierarchy of business component classes. CSSBusComp is at the top of the hierarchy. (Account Category in [Figure 119](#) is in the CSSBusComp business component class.) All other specialized business component classes (CSSBCOppty is an example) are derived from CSSBusComp.

The functionality that is common between business components includes navigation (moving through a result set returned from the database), get or set field values in records, create and delete records, commit changes, undo/redo, bookmark, search, and sort.

NOTE: Do not change the Class property of preconfigured business components.

- **Name.** (Required.) Must be unique among all business components in the repository. All references to the business component are done through its name.
- **No Delete, No Insert, No Update properties.** (Default is FALSE.) If set to TRUE, then you can't do data manipulation operations.
- **Search Specification.** A conditional expression used to restrict the records retrieved.
- **Sort Specification.** A sort expression used to order the records returned.
- **Table.** (Required.) The name of the SQL table from which records are retrieved to populate the majority of fields in the business component. A list of tables appears in a picklist.

Adding Fields to a Business Component

To add a field to a business component

- 1 In the Object Explorer window, select the Business Component object type.
- 2 In the Object List Editor window, select the business component you are defining.
- 3 In the Object Explorer window, expand the Business Component object type, and then select Field.
- 4 Modify the Field properties, as shown in [Figure 120](#).

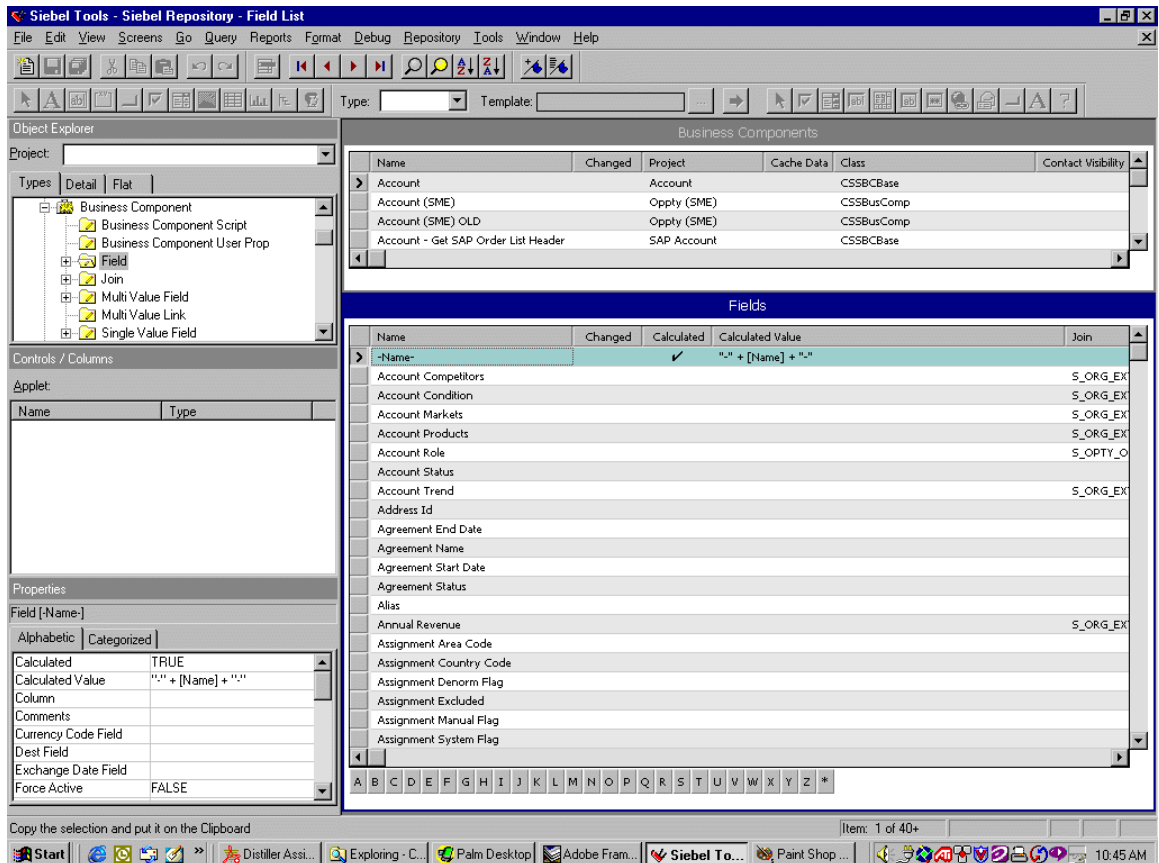


Figure 120. Single Value Field Properties of a Business Component (Account Product)

Field Properties

Following are descriptions for key properties of the Field object type. For a complete list and more detailed descriptions, see *Object Types Reference*.

- **Column.** (Required unless it is a calculated field.) The name of the table's column. Default table is the business component table.
- **Name.** (Required.) User-defined name for the field. It must be unique within the business component.
- **No Copy.** If TRUE, the field's value is not copied into the newly created record during a Copy Record operation. The No Copy property of a Multi Value Link will override the No Copy property of a Field in the child Business Component.
- **Read Only.** If TRUE, the field value cannot be changed by the user.
- **Required.** If TRUE, a value must be entered before the record can be written.
- **Text Length.** Siebel Tools gets the text length from the database and, for columns with a physical type of varchar, sets the Text Length property for a business component field to this value.

If the physical type is character(1), the Text Length is set to 1 and the Type is set to DTYPE_BOOL.

For number, date, and datetime fields, Siebel Tools does not put a value in Text Length. Exceptions are fields mapped to foreign key columns (these columns have names that end in ID and have a physical type of varchar and length of 15). These fields get a Siebel Type of DTYPE_ID and a Text Length of 15.

If you edit the value in the Text Length property, it is ignored unless the value in a picklist is longer than what is specified on the business component field. In this case you get an error.

- **Type.** The field data type.

For information about the data types for the field object type, see *Object Types Reference*.

Single-Value and Multi-Value Fields

As shown in [Figure 121](#), the Object Explorer shows Business Component child object types for Field, Single Value Field, and Multi-Value Field.

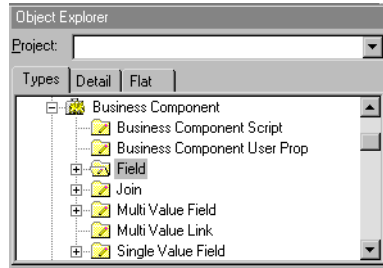


Figure 121. Field, Single Value Field, and Multi-Value Field Object Types

Normally you would modify all Field objects by selecting the Field object type, finding the field you want, and making the changes. If you select the Single Value Field object type, you see only Single Value objects and their properties; the Multi Value Field object type shows only Multi Value objects. You can also configure particular Single Value Field and Multi Value Field objects by viewing and changing them within Single Value Field and Multi Value Field object type. Changes you make there will be reflected when you select the changed object in Field object type.

NOTE: You can configure queries to be case sensitive or case insensitive by setting the Use Default Sensitivity property in the configuration (.cfg) file. Set the property to TRUE for case-sensitive queries. Set the property to FALSE for case-insensitive queries. However, you cannot configure fields of the type DTYPE_ID in this way because these fields always conduct case sensitive searches.

Configuring Dual Currency Support

Use the following steps to configure the display of dual currencies.

- 1 Create a new field in the business component to hold the currency code to which the conversion should be performed.
 - This field is not a foreign key to another table; it must be of type `DTYPE_TEXT`.
 - Specify `PickList = PickList Currency`.
 - In the corresponding Pick Map, associate `Pick List Field = Currency Code` with the newly created currency code field.
- 2 Create a new field in the business component to hold the converted currency amount.

NOTE: You cannot configure Forecast business components to display dual currency because the list columns displaying monetary values do not map to fields. The list columns display values that are computed by buttons using specialized methods.

- This field must be of type `DTYPE_CURRENCY`.
 - It must be a calculated field, `Calculated Value = [Unconverted Amount]`. The field `Unconverted Amount` must also be of type `DTYPE_CURRENCY`.
 - The `Exchange Date Field` property must point to a field of type `DTYPE_DATETIME`.
 - The `Currency Code Field` property points to the currency code field of [Step 1](#).
- 3 Set the Runtime property to be `TRUE` in the applet that displays the converted currency.

A pick or detail applet need not be specified, because your Siebel application automatically launches the default applet that matches the field type.
 - 4 Before that currency conversion takes place, the underlying currency business component must be filled with a minimum number of valid values. To access the lists of currencies, conversion dates, and exchange rates in your Siebel application, navigate to `Site Map > Application Administration > Currencies`.

- The two currencies between which you want to convert must be marked as active (for example, name the original one O and the converted one C).
- At least one exchange rate value must be defined for O to C. (For the reverse conversion C to O, another exchange rate value is required.)
- At least one of the exchange rates of a certain exchange direction must have a date at or before the date that is used as 'Exchange Date'.

The following is an example of configuring dual currency display.

To configure dual currency display

- 1 Add a field to the Opportunity business component for the currency code to which the conversion is made, as shown in the following table:

Property	Value
Name	My_Currency
Type	DTYPE_TEXT
Join	S_OPTY_X
Column	ATTRIB_03
PickList	PickList Currency

The field is stored in an unused column in the extension table S_OPTY_X.

- 2 Add a record to the field's child Pick Map object, as shown in the following table:

Property	Value
Field	My_Currency
Pick List Field	Currency Code

- 3 Add a field to the Opportunity business component for the converted revenue, as shown in the following table:

Property	Value
Name	My_Cvt_Revenue
Calculated	TRUE
Calculated Value	[Revenue]
Currency Code Field	My_Currency
Exchange Date Field	Sales Stage Date
Type	DTYPE_CURRENCY

- 4 Add two new list columns to the Opportunity List Applet, as shown in the following tables:

Property	Value
Field	My_Currency
Display Name	Converted Currency Code

Property	Value
Field	My_Cvt_Revenue
Display Name	Converted Revenue
Runtime	TRUE

- 5 Compile the Oppty and Oppty (SSE) projects.

Creating or Modifying a Business Object Definition

General instructions for modifying or creating object definitions are provided in “Modifying, Copying, and Creating New Object Definitions” on page 117.

Figure 122 shows the object definition for a business object called Account.

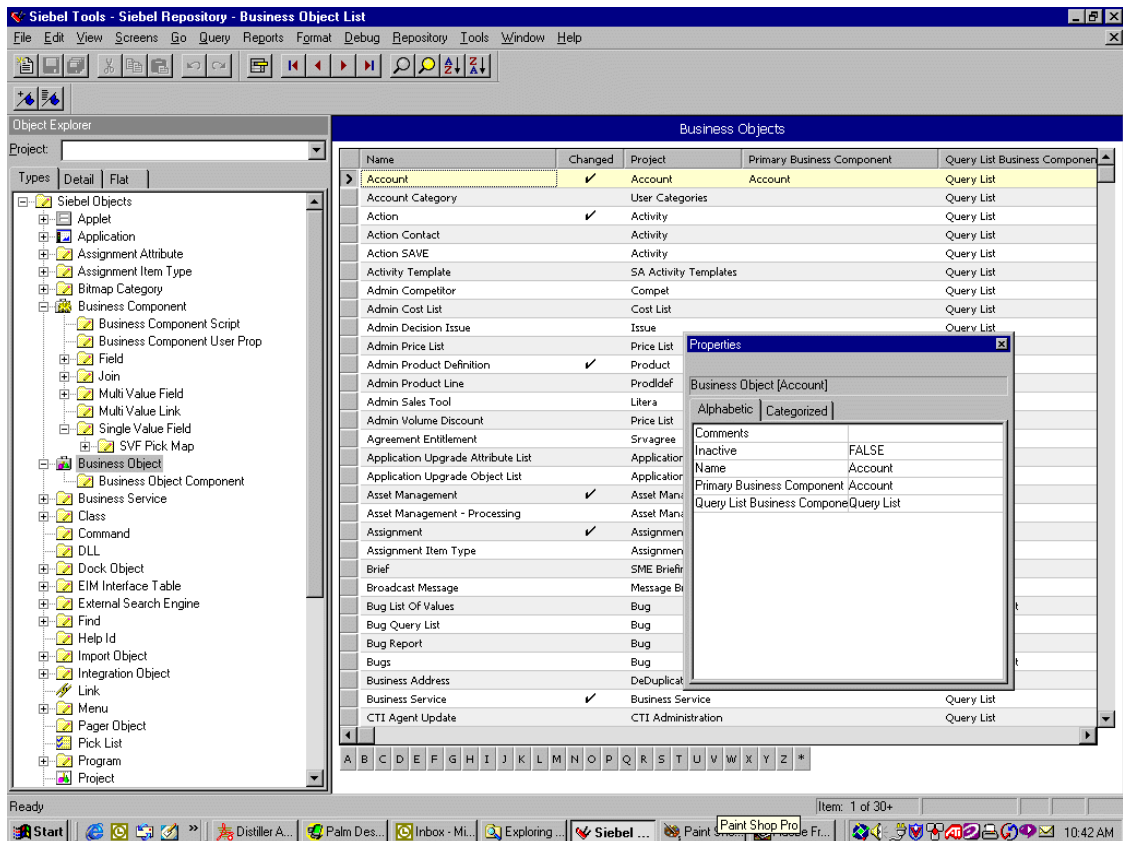


Figure 122. Properties of the Account Business Object

Business Object Properties

Following are descriptions for key properties of the Business Object object type. For a complete list and more detailed descriptions, see *Object Types Reference*.

- **Name.** The name of the business object must be unique among business objects in the repository. All references to the business object are done through its name.
- **Query List Business Component.** The default value is Query List. It identifies the business component used to store predefined queries for the business object.

Mapping Business Components to Business Objects

As shown in [Figure 123](#), select Business Object Component under Business Object, and then pick from the picklist the business component that the business object needs to be associated with.

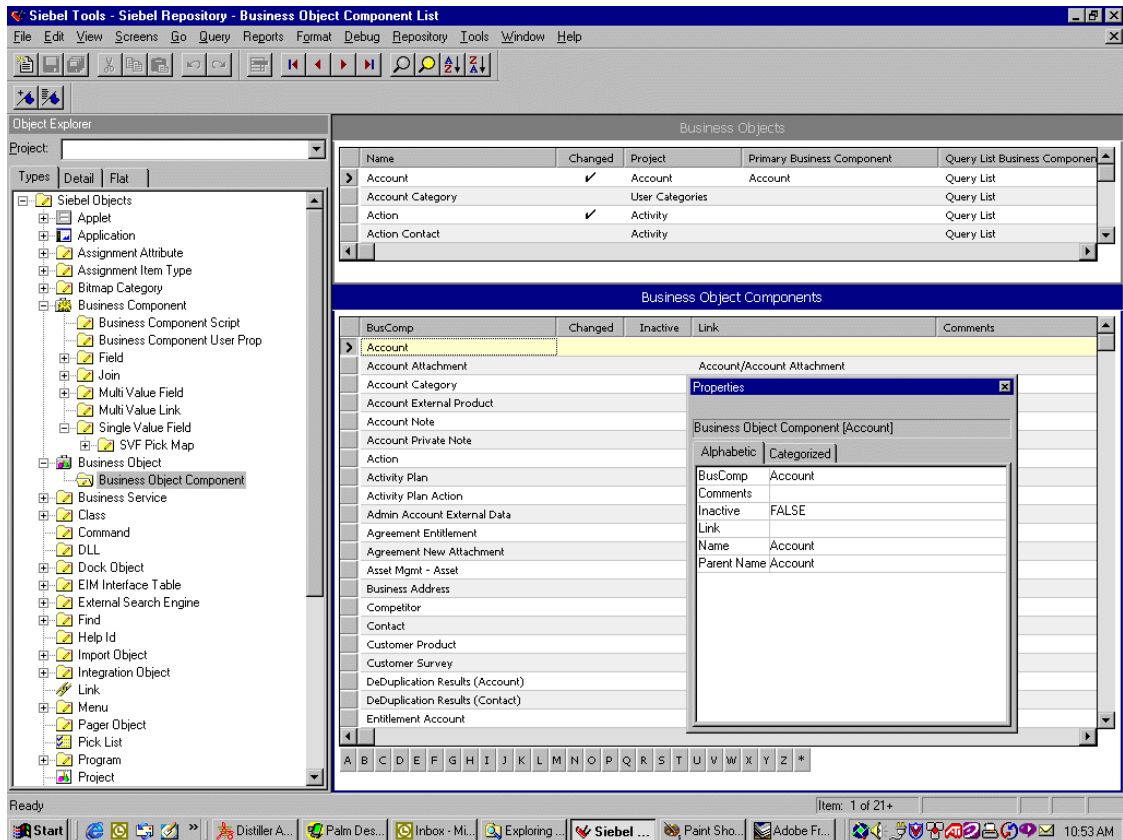


Figure 123. Mapping a Business Object (Account) with a Business Component (Account)

Mapping Business Objects to Data Objects

Figure 124 illustrates object definition mappings.

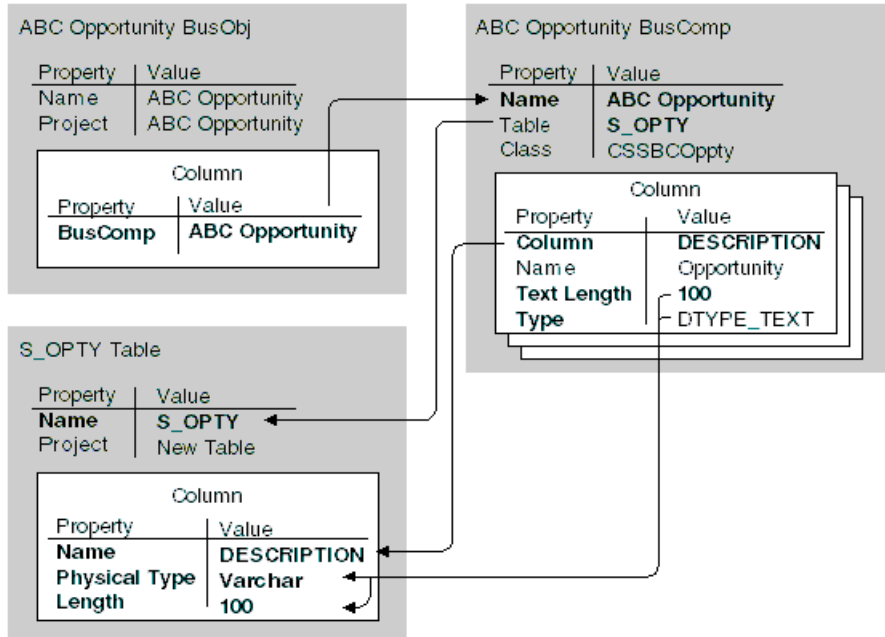


Figure 124. Mappings Between Business Objects and Data Objects

If you change the value of the Name property of an object definition, you must update property values in other object definitions that refer to the original Name value so that they now refer to the new Name value.

Mapping User Interface Objects to Business Objects

Figure 125 shows how objects in the User Interface Layer are mapped to objects in the Business Objects Layer.

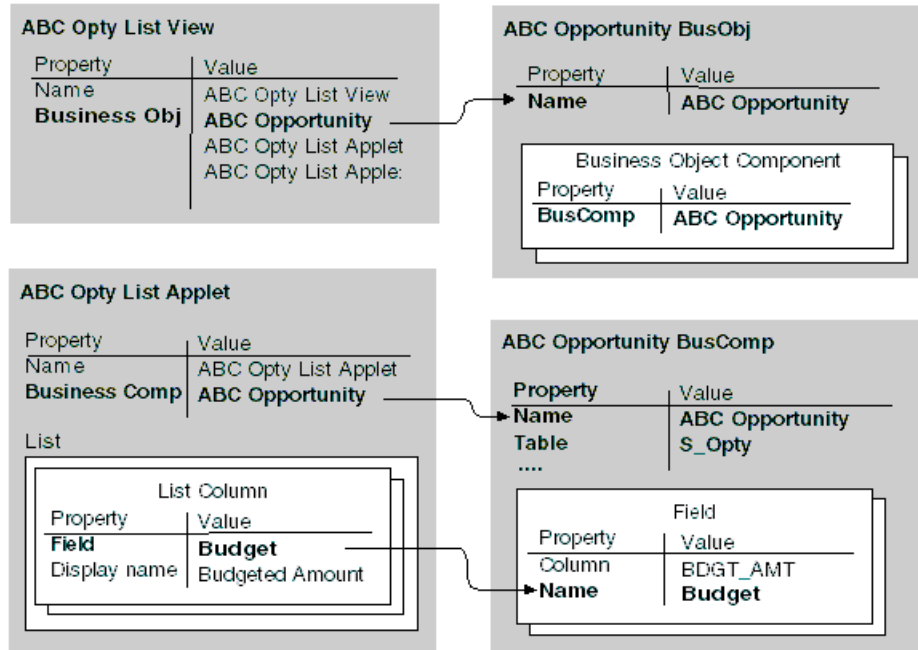


Figure 125. Mapping of UI Layer Definitions to BO Layer Definitions

Using Wizards to Create Objects

Several wizards are available for creating business object layer objects. The following section describes the Business Component Wizard, OLEDB Rowset Wizard, and Integration Object Wizard.

Business Component Wizard

You use the New Business Component Wizard to create Business Component objects.

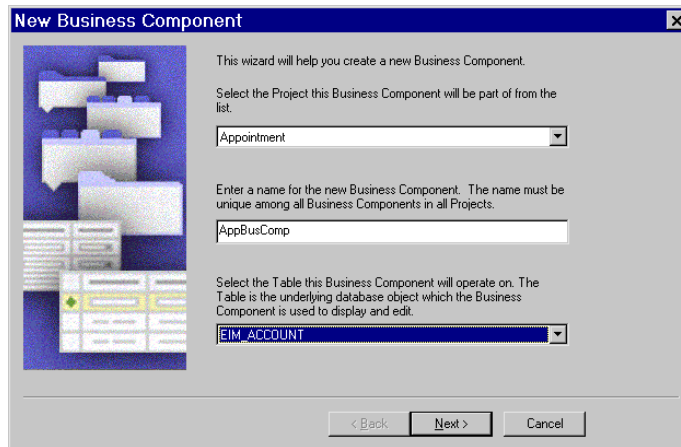
Defining Business Objects and Business Components

Using Wizards to Create Objects

To open the Business Component wizard

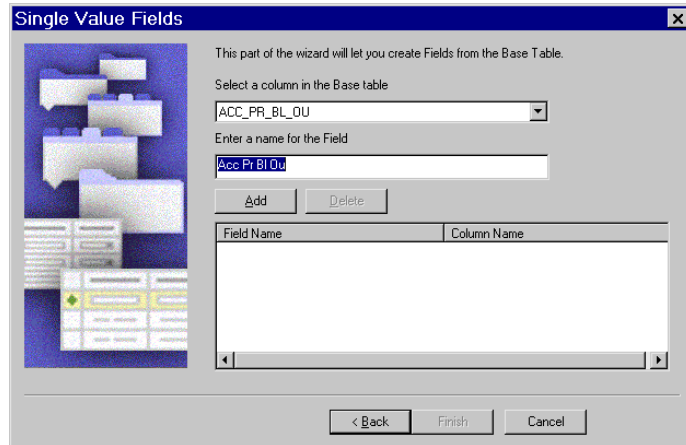
- 1** Choose File > New Objects.
- 2** Select the BusComp icon, and click OK.

The New Business Component dialog box appears as shown in the following figure.



- 3 Select a Project and the master Business Component and click Next.

The second New Business Component dialog box appears.



- 4 Select a column in the Base table and enter a name for the field.
- 5 Click Add and Finish.

When you click Finish, you are taken to the business component you just created in the Object List Editor, where you can further configure the new Business Component object.

For more information about business components, see [“Business Components” on page 342](#).

OLEDB Rowset Wizard

OLEDB is a specification for a set of data access interfaces designed to enable heterogeneous datastores to work together. Components built to the OLEDB standard behave as a table, even though complex computing processes can occur between the data sources and the applications.

The Siebel OLEDB Rowset wizard is a read-only provider that exposes Siebel business components as virtual OLEDB tables. Using the Siebel OLEDB Provider, external OLEDB-enabled applications can access data stored in Siebel by referring to Siebel objects like Contact or Account without the need to understand the internal functioning of the Siebel Data Model. You can configure the Siebel business components that are exposed to the client application as OLEDB tables.

The OLEDB Rowset wizard steps you through the process of creating OLEDB tables.

For more information about the OLEDB Rowset wizard, see *Transports and Interfaces: Siebel eBusiness Application Integration Volume III*.

Integration Object Wizard

You can create Integration Object objects using the Integration object wizard. For more information about integration objects and Integration Object Wizard, see *Integration Platform Technologies: Siebel eBusiness Application Integration Volume II*.

Logical User Interface Objects Layer

9

This chapter describes objects in the User Interface Objects Layer in the Siebel application architecture.

Major User Interface Object Types

Figure 126 shows user interface objects in a typical Siebel application session.

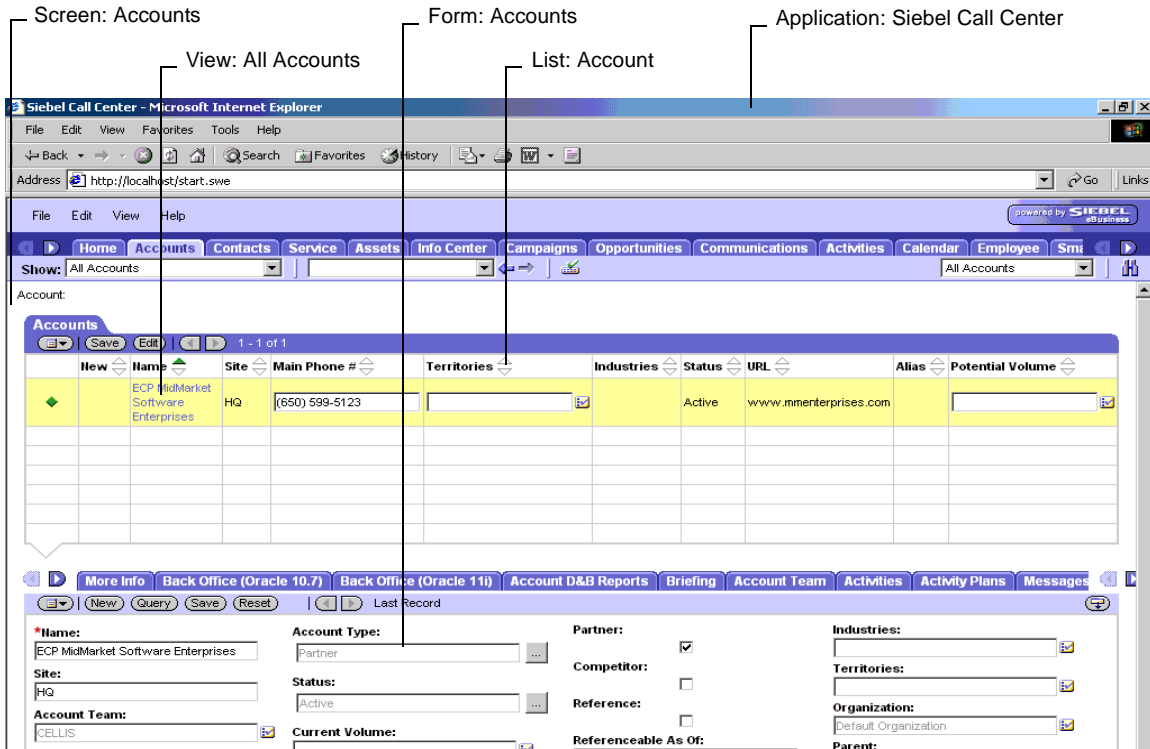


Figure 126. User Interface Objects in a Typical Siebel Application Session

The Application, Screen, View, and Applet object types have hierarchical (one-to-many) relationship based on parent/child object type relationships and property settings. The full set of user interface object types and their relationships is illustrated in [Figure 127 on page 450](#).

Logical User Interface Objects Layer

Major User Interface Object Types

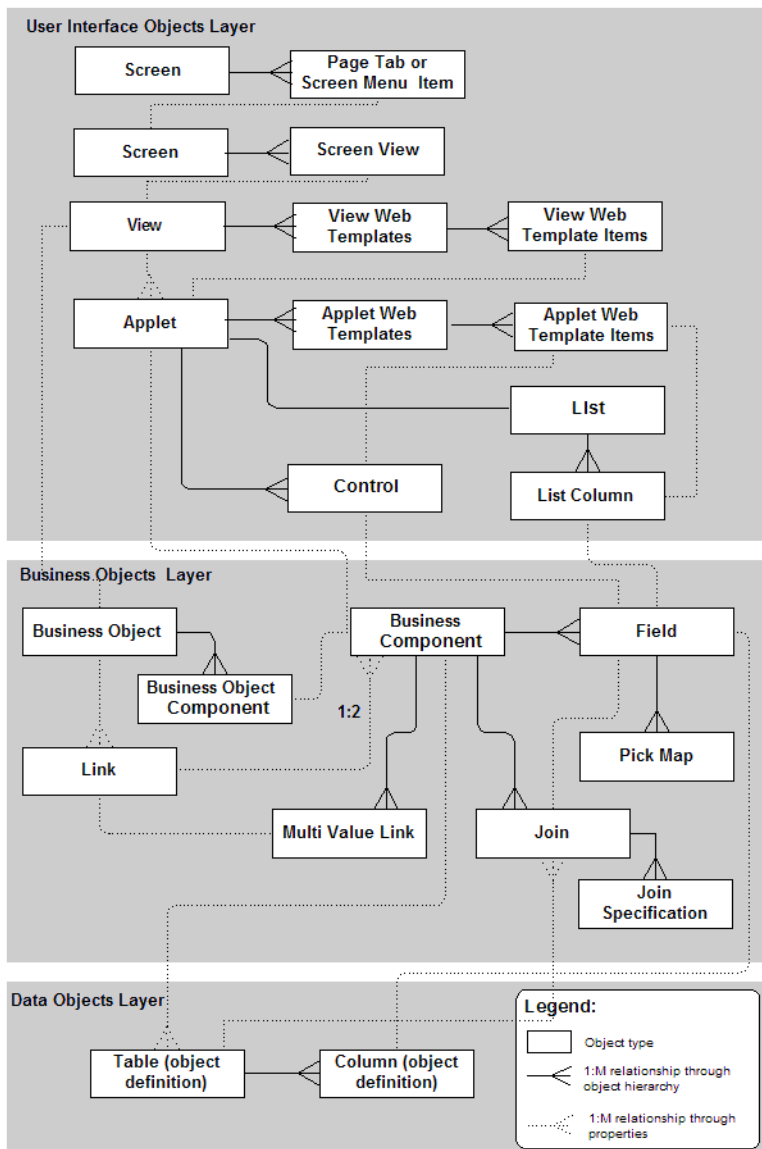


Figure 127. Details of User Interface Architecture

The following user interface object types are introduced in [Figure 127 on page 450](#):

The object types in the diagram (plus some additional ones that are not shown) are described briefly below. For more detailed descriptions, see *Object Types Reference*. Note that these are only a subset of the full set of object types in the Siebel architecture.

Application. An application is a collection of screens. The application is opened in a Web browser on the user's desktop by attaching to a specified URL. The screens are accessed from the tab bar and the Site Map (Screen Menu Item), as defined in the application. Siebel eService is an example of an application. Each combination of screens that is appropriate to a specific class of users can be provided as an application.

Page Tab. A page tab object definition associates a screen to the page tab's parent application object definition and includes it as a tab in the tab bar.

Screen Menu Item. A screen menu item object definition associates a screen with the application and includes the screen as a hyperlink on the Site Map.

Screen. A screen is a logical collection of views. It is not a visual construct in itself; rather, it is a collection of views that the screen tabs and view tabs can display. The active screen is selected from the Site Map or the screen tabs.

NOTE: For more information, see [“Screens” on page 502](#).

Screen View. A screen view object definition associates a view with the screen view's parent screen object definition. This is how views are included in screens.

View. A view is a collection of applets which appear onscreen at the same time. A view can be thought of as a single window's worth of related data forms (applets). Generally, the Siebel application window displays one view at any one time. The user can select the current (active) view from the view bar, thread bar or from a hyperlink on the Site Map. A view is associated with the data and relationships in a single business object.

Applet. An applet is a form, composed of controls, that occupies a portion of the Siebel application window. An applet can be configured to allow data entry, provide a table of business component records, or display business graphics, a navigation tree, or a similar user interface unit. It provides viewing, entry, modification, and navigation capabilities for data in one business component. Pop-up windows for multi-value groups and record selection are also implemented as applets.

Control. One control object definition corresponds to one data control or a button in a form applet, such as a text box, check box or command button. A control is something in the applet with which the user can interact. A control usually either exposes data from one field in the business component, or invokes programming logic (in the case of a PushButton control).

For more information, refer to [“Form Applet Controls” on page 465](#).

List. List is a child object type of Applet. A list object definition specifies property values that pertain to the entire scrolling list table and provides a parent object definition for a set of list columns.

NOTE: For more information, refer to [“Configuring the List” on page 476](#).

List Column. A list column object definition corresponds to one "column" in the scrolling list table in a list applet, and to one field in the business component.

NOTE: For more information, refer to [“Configuring the List Columns” on page 476](#).

Web Template, Applet Web Template, View Web Template. Identify external HTML (or other markup language) files that define the layout and Siebel Web Engine interactions for an applet or view.

Applet Web Template Item. Defines list columns and controls to be mapped to place holders in a Web Template. They contain the name of the list column or control as well as a unique identifier for a placeholder in the template. The placeholder to which an Applet Web Template Item is mapped determines the position of the item within the template. Applet Web Template Item objects are automatically created when you drag and drop controls and list columns onto a Web template using the Web Applet Layout Editor.

View Web Template Item. Defines applets to be mapped to place holders in a Web Template. They contain the name of the applet as well as a unique identifier for a placeholder in the template. The placeholder to which a View Web Template Item is mapped determines the position of the applet within the template. View Web Template Item objects are automatically created when you drag and drop applets onto a Web template using the Web Applet Layout Editor.

Applets

An *applet* is a data entry form, composed of controls, that occupies some portion of the Siebel application user interface. An applet can be configured to allow data entry, provide a table of data rows, or 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 a business component. Although the same business component can be associated with multiple applets, an applet is associated with only one business component.

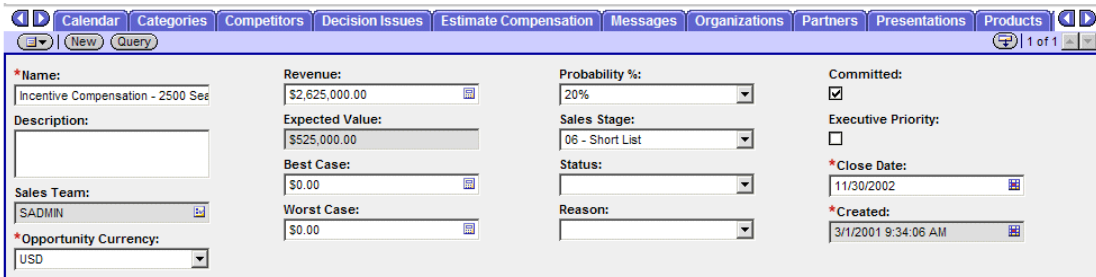
Applets are associated with one or more Siebel Web templates. Web templates are files that contain HTML and proprietary Siebel tags that define the layout and format of the applet in the user interface.

For more information about applets and Web templates, see [Chapter 15](#).

Types of Applets

This section summarizes the various applet styles found in Siebel applications.

- **Form applet.** A *form applet* displays data in a data entry form. Fields in the business component appear on the form applet as text boxes, check boxes, and other standard controls. A form applet appears in [Figure 128](#).



The screenshot shows a Siebel CRM form titled "Incentive Compensation - 2500 See". The form is organized into several columns and rows of fields. At the top, there is a navigation bar with tabs for "Calendar", "Categories", "Competitors", "Decision Issues", "Estimate Compensation", "Messages", "Organizations", "Partners", "Presentations", and "Products". Below the navigation bar, there are buttons for "New" and "Query", and a status indicator "1 of 1". The form fields are as follows:

Field Name	Value
*Name:	Incentive Compensation - 2500 See
Revenue:	\$2,625,000.00
Expected Value:	\$525,000.00
Best Case:	\$0.00
Worst Case:	\$0.00
Opportunity Currency:	USD
Probability %:	20%
Sales Stage:	06 - Short List
Status:	
Reason:	
Committed:	<input checked="" type="checkbox"/>
Executive Priority:	<input type="checkbox"/>
*Close Date:	11/30/2002
*Created:	3/1/2001 9:34:06 AM

Figure 128. Opportunity Form Applet

See [“Form Applets” on page 462](#) for more information about form applets.

- List applet.** A *list applet* allows the simultaneous display of data from multiple records. A list applet displays data in a list table format, much like a spreadsheet or word processor table. Rows in the list applet correspond to records in the business component; list columns in the list applet correspond to fields in the business component. In addition to textual data, lists also support images in JPEG and GIF formats and edit controls such as check boxes, drop-down lists, MVGs, and text fields. A list applet appears in [Figure 129](#).

See “List Applets” on page 472.

Name	Site	Main Phone #	Territories	Industries	Status	URL
3COM	Santa Clara	(408) 326-5000		manufacturing industries	Gold	www.parker.com
3Com Corporation	Headquarters	(415) 329-6500		manufacturing industries	Active	
3Com Services	US	(415) 329-6500		manufactured hardware (general)	Active	
AMCO Communications	San Rafael	(415) 491-2300		steel pipe & tubes	Active	
Acme Inc.	HQ	(510) 245-3000		advertising agencies	Active	
Acme New England Division	Boston	(617) 232-1121		animal specialties	Active	
*	Alliance Program					

Figure 129. List Applet

- Pick applet.** A *pick applet* is a dialog box window that appears when a selection is to be made in a control or list column that has the check mark icon to its right. The pick applet provides a list or table of selection values, from which the user selects a value or record. A pick applet displays data that has a M:1 relationship to the data in the parent applet. A pick applet appears in [Figure 130](#).

For more information about pick applets, see “[Pick Applets and Static Picklists](#)” on page 613.

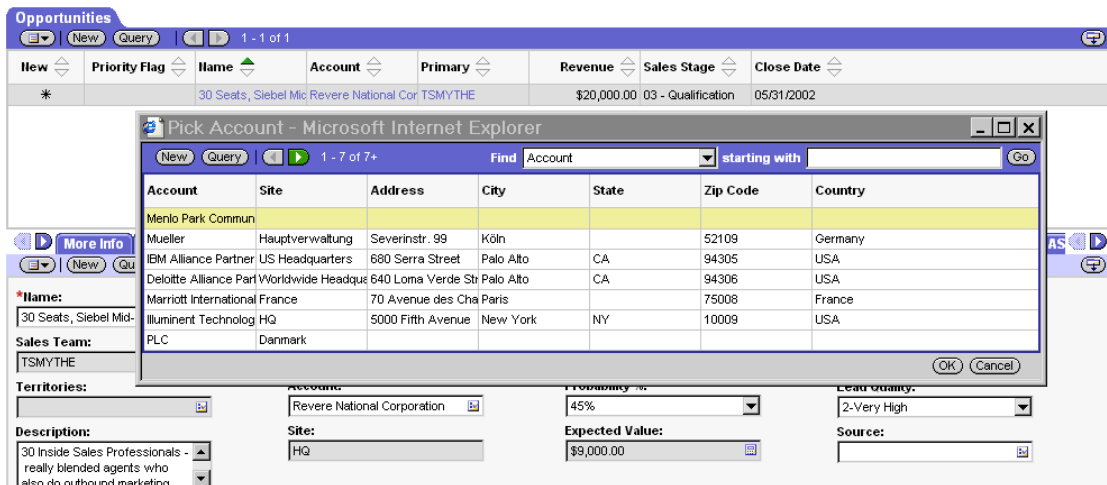


Figure 130. Pick Applet

- Multi-value group applet.** A *multi-value group applet* is used for entry, maintenance, and viewing of a list of detail records associated with one or more fields in the currently displayed master record. MVGs allow the user to associate multiple records to a single field in a form or list and provide a way of representing one-to-many relationships within a single record of data.

There are two ways in which users can add data to the MVG:

- Inputting data

- Selecting from existing database records

For example, an account can have multiple members on an account team. A multi-value group applet can be invoked from the Account Team check mark control in the Account Entry form applet to display or maintain the list of members associated with the current account.

Both Pick List and MVG dialog boxes allow users to add or edit records in their lists, provided the user has permission to do so. A multi-value group applet appears in [Figure 131](#).

For more information about MVGs, see [“Multi-Value Group and Association Applets”](#) on page 687.

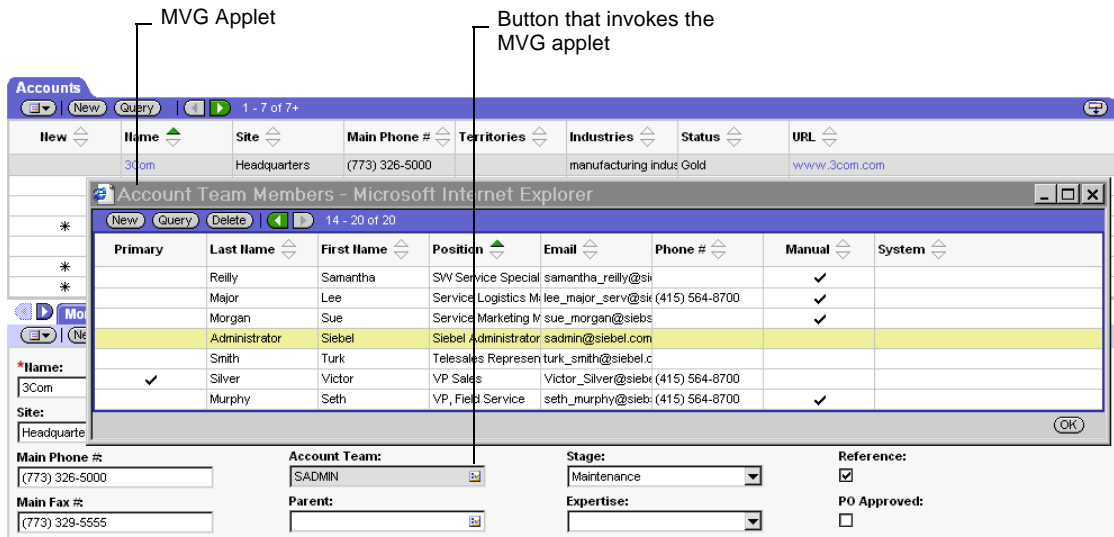


Figure 131. Multi-Value Group Applet

- Chart applet.** A *chart applet* graphically displays data from a business component in a bar chart, line graph, pie chart, scatter diagram or other format. It summarizes and illustrates data relationships. Charts are usually accessed through a tab in the third-level navigation level and contain a number of sub-category views (multiple charts of data). These are displayed in an overview of miniature chart graphics (.gif images) with title text. Both the mini-graphics and the title text for the chart are hyperlinked to the detailed version of the chart. A chart applet appears in [Figure 132 on page 458](#).

For more information about chart applets, see [Chapter 13, “Special-Purpose Applets and Controls.”](#)

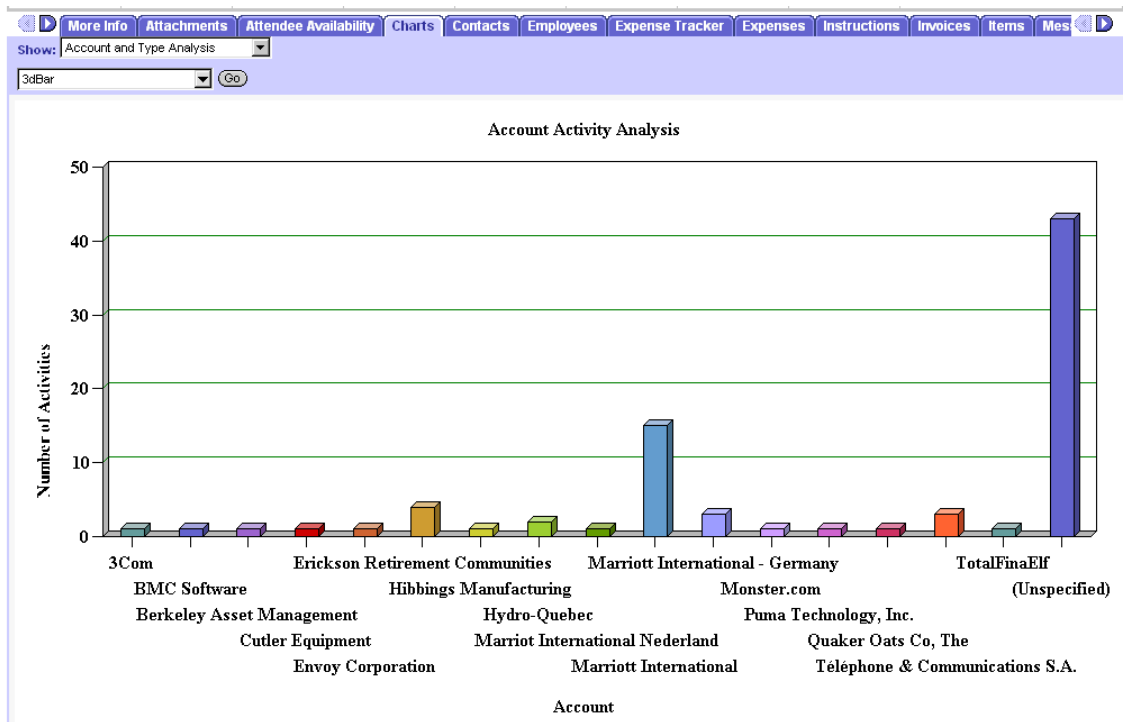


Figure 132. Chart Applet

Once the full chart is displayed, the fourth-level navigation Show drop-down list is used to navigate between the various charts of data (for example, Sales Pipeline or Size Analysis). Additional controls on the graphs provide a means to filter the data being charted and to change the chart type (from bar to line, for example). See [Figure 132 on page 458](#).

- **Association applet.** An *association applet* provides the user with the ability to associate records of two business components that have a many-to-many relationship. It is invoked from the check mark icon in a multi-value group applet. An association applet appears in [Figure 133 on page 459](#).

For more information about association applets, see [Chapter 12, “Multi-Value Group and Association Applets.”](#)

Last Name	First Name	Mr./Ms.	Account	Job Title	Work Phone #
Adams	Al		Aqua Inc.		
Adams	Betty		Aqua Inc.	Bus Conslnt	(613) 555-4707
Adams	Jamie		Marriott International	VP, Marketing	(310) 455-7664
Aitken	Mary	Ms	Kellogg Company	Program Mgr, Siebel	(616) 555-6344
Alkana	Wanda	Mrs.	Bell Canada	Mgr	(514) 555-2504
Aker	Charles	Mr.	Manulife Financial	Inforte	(312) 555-0900 x2202
Alacon	Terry	Ms	Tower Industries	Manager, IT	(415) 555-7834

Figure 133. Association Applet

- **Explorer or Tree applet.** A *tree applet* is used to create an explorer view that allows the user to navigate hierarchically through a structured list of object instances. A tree applet appears in [Figure 134](#).

For more information about tree applets, see “[Special-Purpose Applets and Controls](#)” on page 717.

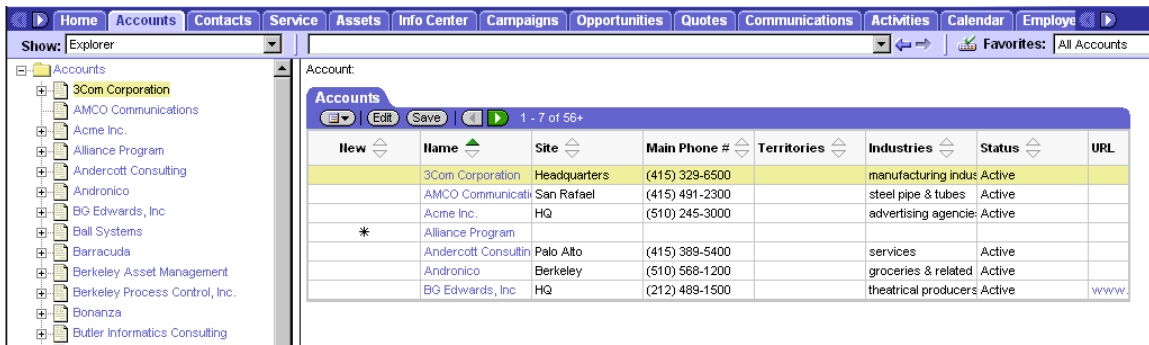


Figure 134. Explorer or Tree Applet

- File attachment applet.** *File attachment applets* provide access to external documents, such as spreadsheets, word processing documents, and presentations, that have been imported in compressed format into records in a Siebel application. A file attachment applet appears in [Figure 135](#).

For more information about chart applets, see [Chapter 13, “Special-Purpose Applets and Controls.”](#)

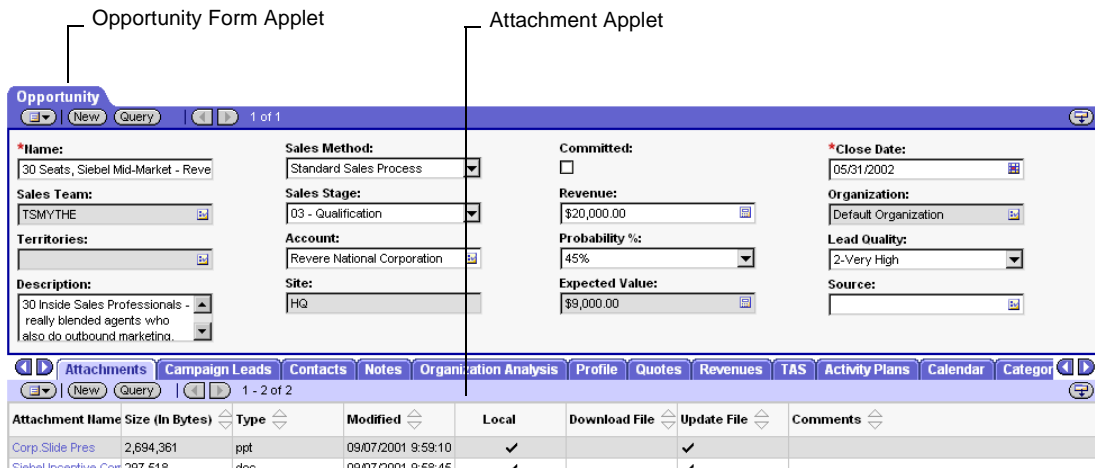


Figure 135. File Attachment Applet

Form Applets

A *form applet* presents business component information in a data entry form layout. An example of a form applet in Siebel Call Center appears in [Figure 136](#).

The screenshot displays a Siebel Call Center form applet with a navigation bar at the top containing tabs for Calendar, Categories, Competitors, Decision Issues, Estimate Compensation, Messages, Organizations, Partners, Presentations, and Products. Below the navigation bar are buttons for 'New' and 'Query', and a status indicator '1 of 1'. The form itself is organized into several sections:

- Name:** Incentive Compensation - 2500 See
- Description:** (Empty text area)
- Sales Team:** SADMIN
- *Opportunity Currency:** USD
- Revenue:** \$2,625,000.00
- Expected Value:** \$525,000.00
- Best Case:** \$0.00
- Worst Case:** \$0.00
- Probability %:** 20%
- Sales Stage:** 06 - Short List
- Status:** (Empty dropdown)
- Reason:** (Empty dropdown)
- Committed:**
- Executive Priority:**
- *Close Date:** 11/30/2002
- *Created:** 3/1/2001 9:34:06 AM

Figure 136. Form Applet in Siebel Call Center

Users enter data using text boxes, check boxes, buttons, and similar visual constructs known collectively as *controls*. The Control object type implements data controls, as well as buttons and links that invoke methods. The controls that display data show a value from a particular field in the current business component row. The relationships between objects used to implement a form applet appear in [Figure 137](#).

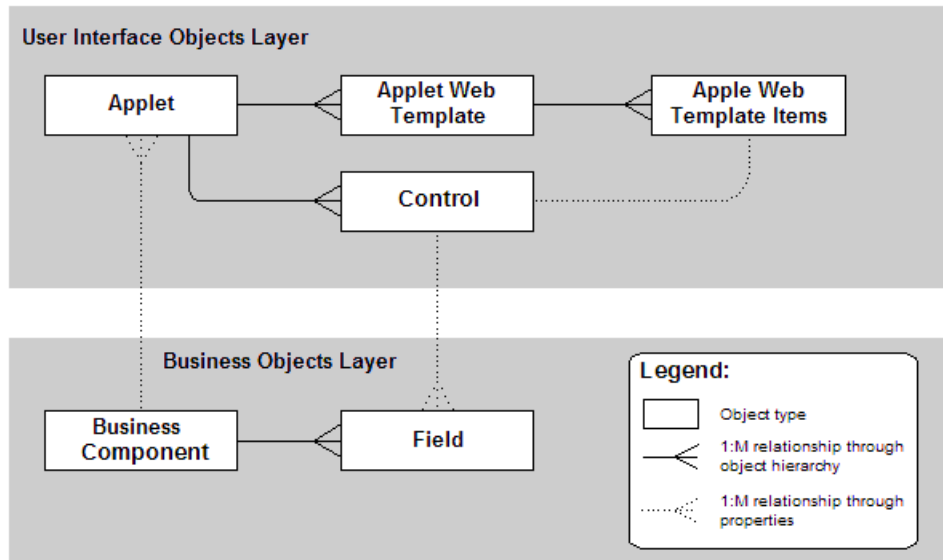


Figure 137. Form Applet Architecture

A form applet is implemented in Siebel Tools by means of an applet object definition, multiple control object definitions, applet Web template definitions, and applet Web template item definitions. These object types have the following roles:

- **Applet.** Provides the properties that belong to the entire applet, such as the Name, Business Component, and specifies in the Class property that the applet is a form applet (CSSFrame class).
- **Control.** Identifies individual data controls in the data entry form, such as text boxes and command buttons. A control either corresponds to one field in the business component, or invokes program logic (in the case of a button control).

- **Applet Web Template.** Associates an applet to a Web template. Web templates determine the layout and format of the applet when it is rendered in the user interface. An applet can be displayed in four of the five standard modes. An Applet Web template is defined for each mode. The modes are listed below:

- **Base.** Read only.
- **Edit.** Used for editing records where users can update values. You can also use the edit for creating new records and querying.

NOTE: Typically, New and Query modes are not necessary because Edit mode can be used for these type of actions.

- **New.** Used for creating a new record where the requirements for new mode are different from the edit mode.
- **Query.** Used for querying where the requirements for the query mode are different from the edit mode.
- **Edit List.** Not applicable for form applets.
- **Applet Web Template Items.** Maps controls to placeholders tags in a Web template. It contains the name of a control or list column as well as unique identifier of a template placeholder. The placeholder determines its position of the control or list column in the Web page rendered at runtime. Applet Web Template Items are automatically populated when users drag and drop controls into placeholders using the Applet Web Layout editor.

For more information about mapping controls and list columns to Web templates, see [“Editing the Web Layout of Applets” on page 544](#).

Form Applet Controls

Following creation and placement of data controls in the applet using the Applet Web Layout Editor, the properties of the corresponding control object definitions can be edited in the Object List Editor.

NOTE: Controls for form applets using the “Applet Form 4 Column (Edit/New)” Web template can be associated to either a “2-Column Wide field” or a “1-Column Wide field”. To associate a control to a “2-Column Wide field”, you must set the HTML Width property to 412. If you do not specify an HTML Width property, the control will appear as a “1-Column Wide field” even when it is associated to a “2-Column Wide field” on a form applet.

All control object styles have the following important properties, except where otherwise noted:

- **Name.** The name of the control, for identification by other objects.
- **HTML Type.** Specifies the style of the control. Examples of control types include Field, Text, ComboBox, CheckBox, Button, and Active X. These Control styles are described in a subsection below.

See the sections below for descriptions of some controls.

NOTE: HTML Type values in the repository that begin with the prefix “cfg” are used for Siebel Product Configurator.

- **Caption.** The Caption property provides descriptive text in, on, or near the control, depending on the type of the control. The caption specified for a button control is the text that appears on the button face. The Caption property is unspecified for control styles which do not have identifying text. An example of a Text control with a caption is shown in [Figure 138](#).

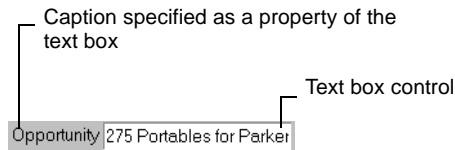


Figure 138. Caption for a Text Control

- **HTML Sequence.** An integer value specifying the tab sequence of this control relative to other controls in the applet. Tab sequence is the order in which the tab key moves the focus from control to control on the applet. The lower the integer value you enter for sequence in a particular control in the applet, the sooner the tab key will access this control relative to others.

Some control styles (as specified in the Type property) are described briefly in the following sections.

Text Controls

A *text control* displays text inside a rectangular box. An example of a text control is Name, shown in [Figure 139](#).



Figure 139. Text Control

Some characteristics of Text controls are as follows:

- A text control allows the entry and editing of text, unless the Text control is read-only (in which case it has a gray background, and displays text which cannot be altered).

- A text control displays data of a particular data type, such as alphanumeric, numeric, date, or currency.
- A Select icon is automatically attached to the right edge of a Text control when the MVG Applet property has a non-blank value or the Pop-up Edit property is TRUE. This enables the user to call up a multi-value group applet or a calendar or calculator widget.

NOTE: The Runtime property must also be TRUE any time the field is supposed to pop up a Calendar or Calculator type control.

- The Select button is attached to the right edge of a text control when the Pick Applet property has a non-blank value. This allows the user to call up a picklist by clicking the icon.

Static picklists and pick applets are discussed in [Chapter 11, “Pick Applets and Static Picklists.”](#)

NOTE: Trailing spaces are truncated in data displayed through the Siebel application user interface or through Siebel Tools.

Controls of type Text have the following essential properties:

- **Field.** The field in the business component from which the Text control displays data.
- **Display Format.** A format specification for data displayed by the Text control, used for numeric, date, currency, and similar non-text data types. Used as follows:
 - For DTYPE_NUMBER data, the property can be left blank (indicating that the appearance of numeric values should be as set in the Regional Settings section of the Windows Control Panel) or explicitly specified using 0, #, +, minus sign, comma, and period symbols.
 - For DTYPE_CURRENCY data, the property can be specified explicitly using the same symbols as for DTYPE_NUMBER, plus the dollar sign. The display of currency values can also be controlled using the Scale field in the Currencies view under the Application Administration screen.

- For DTYPE_DATETIME data, one of the keywords Date, Date Time, Date TimeNoSec, and TimeNoSec may be specified.
- For DTYPE_DATE data, the property can be left blank (indicating that the appearance of date values should be as set in the Windows Control Panel) or explicitly specified using combinations of M, D, Y, and / symbols.
- For DTYPE_TIME data, the keyword TimeNoSec can be entered, the property may be left blank (indicating that the appearance of time values should be as set in the Windows Control Panel), or a format mask may be explicitly specified using combinations of H, h, m, s, and : symbols.
- For DTYPE_PHONE data, the Display Format property is left blank, and the Windows Control Panel setting is used.

NOTE: Postal code formatting options are not explicitly provided, and hyphens in a postal code are not supported. Generally, for postal codes you should use the DTYPE_NUMBER data type, and a format mask in the Display Format property consisting of number signs and blank spaces, such as ##### #### for U.S. Zip + 4 postal codes.

- **Read Only.** A TRUE/FALSE value. Indicates if the user can edit the value displayed in the text box.

NOTE: The Read Only property must be set to FALSE to use the Runtime property to access multi-value groups and pick applets.

- **Runtime.** This is a TRUE/FALSE value. When the text box control has an MVG Applet or Pick Applet property setting other than blank, a value of TRUE in the Runtime property directs the system to activate an icon or drop-down arrow to the right of the text box. A FALSE value directs the system not to provide the icon or arrow. This makes the multi-value group or pick applet inaccessible.

NOTE: A Runtime setting of TRUE, combined with blank MVG Applet and Pick Applet property settings, directs the system to determine from the data type of the underlying field if an icon for a calculator, calendar, or currency pop-up applet should be provided.

- **MVG Applet.** Identifies the applet to use for the multi-value group dialog box (multi-value group applet). The field for the control must be a multi-value field, and the Runtime property must be set to TRUE.
- **Pick Applet.** Identifies the applet to use for the picklist dialog box (pick applet). The field for the control must have a picklist specified, and the Runtime property must be set to TRUE.

Button Controls

Button controls initiate an action when clicked. There are several types of button controls, including Button, MiniButtonEdit, and MiniButtonNew. The Button control type is rendered as an HTML button. MiniButtons are custom controls whose formatting is defined in a .swf file. All button types are formatted slightly different. Example buttons are illustrated in [Figure 140](#).

For more information on customized controls, see “[Creating Custom HTML Control Types](#)” on page 903.



Figure 140. Button Controls

A button can invoke a built-in method (supplied with Siebel applications), or a custom method programmed in Siebel VB, Siebel eScript, or Browser Script.

The Method Invoked property is the name of the method invoked when the button control is clicked.

There are instances when you might want to put your own custom methods in the Method Invoked property. For example, this is the only way to invoke Siebel VB, Siebel eScript, or Browser Script on a button-click event.

NOTE: The Runtime property must be set to TRUE for button controls. Otherwise the method associated with it will not execute.

Combo Box Controls

A *combo box* is implemented as a control with a Type property setting of ComboBox. It consists of a text box with a drop-down button attached at the right edge. The user clicks the drop-down button, which activates a selection list, and then clicks a selection in the list. The selected value replaces the previous value in the box. An example of a combo box is shown in [Figure 141](#).



Figure 141. Combo Box Control

Combo box controls implement special-purpose picklists in chart, calendar, and pick applets. In chart applets they implement the Show and By combo boxes. In calendar applets they implement the user name combo box. In pick applets they implement the Find combo box. Combo box controls appear and behave almost identically to static picklists, but they are implemented through a different control type (ComboBox rather than Text Box).

For information on the use and configuration of the specialized combo boxes in a chart applet and information on static picklists, refer to [Chapter 11, “Pick Applets and Static Picklists.”](#)

Check Box Controls

A *check box* is implemented as a control with a HTML Type property setting of CheckBox. It is a small, open square into which an X can be inserted or removed by clicking the box. An example of a check box appears in [Figure 142](#).

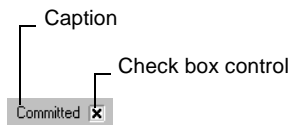


Figure 142. Check Box Control

A check box is used to represent a TRUE/FALSE field with a data type of `DTYPE_BOOL`. A TRUE value is represented as an X, and a FALSE value as an empty box.

ActiveXControl

Allows the placement of an ActiveX control in the applet.

File

Creates a user interface element that can be used to attach a file.

Hidden

Creates an HTML input of type *Hidden*. These controls are not visible in the Web page but can be accessed through scripting.

Link

Used with controls that have an "InvokeMethod" specified (this could be a built-in method that is supplied with Siebel applications). Creates an HTML hyperlink that will invoke the method when activated.

Mailto

Used with controls that contain an email address. The control value will be displayed as a link, which when activated will open the user's default email program with the address filled in with the control value.

Password

Creates a user interface element that can be used to input a password field. The characters entered in this control will be masked by the "*" character.

PositionOnRow

Custom control that shows the currently selected record in a list.

TextArea

Used to create a user interface element that can be used to enter text in multiple lines.

URL

Used with controls that contain URL values. The value will be displayed as a hyperlink, which when activated will take the user to the URL.

Label Controls

A *label control*, is a visual aid only. It has no data display or entry capabilities. Use a label control when you need to place wording somewhere inside the form applet. There are also some specialized label controls, such as the Applet Title.

NOTE: If a caption has any HTML reserved characters, such as &, <, >, “, then it should be HTML encoded as &, <, >, ", respectively.

List Applets

A *list applet* allows simultaneous display of data from multiple records and presents business component information in a list table format with multicolumn layout with each record of data represented in a row. In addition to textual data, lists also support images in JPEG and GIF formats and edits control such as check boxes, drop-down lists, noneditable MVGs, and text fields.

The Accounts List Applet is an example of a list applet in Siebel Call Center, and appears in [Figure 143](#).

List column displaying the contents of the Last Name field and containing drilldown (hypertext) jumps to another view

New	Name	Site	Main Phone #	Territories	Industries	Status	URL
	3COM	Santa Clara	(408) 326-5000		manufacturing industries	Gold	www.parker.com
	3Com Corporation	Headquarters	(415) 329-6500		manufacturing industries	Active	
	3Com Services	US	(415) 329-6500		manufactured hardware (general)	Active	
	AMCO Communications	San Rafael	(415) 491-2300		steel pipe & tubes	Active	
	Acme Inc.	HQ	(510) 245-3000		advertising agencies	Active	
	Acme New England Division	Boston	(617) 232-1121		animal specialties	Active	
*	Alliance Program						

Figure 143. List Applet in Siebel Sales

Columns in a list applet are called *list columns*, and have a corresponding object type (List Column). Data entry in a list applet is performed in the cells that are at the intersections of rows and list columns. Cells in different list columns can function in different ways, depending on the properties of their list columns. Some examples of cell behavior based on list column properties are:

- Cells in some list columns function like text controls in a form applet. This kind of cell is used for the display and editing of a text, numeric, date, or currency value. If the list column is not read-only, you can click the cell to activate an editing cursor, and edit the text.
- Cells in some list columns function like check box controls in a form applet. A check mark in the box is a TRUE value; an empty box has a FALSE value.

NOTE: When TRUE, a check box in a list column holds a check mark symbol, whereas a check box in a control in a form applet holds an X symbol.

- Cells containing underlined, colored text are *drilldown fields*. Drilldown fields let the user navigate from the cell to another view that presents detailed information about the selected row.

The relationships between object types used to implement a list applet appear in Figure 144.

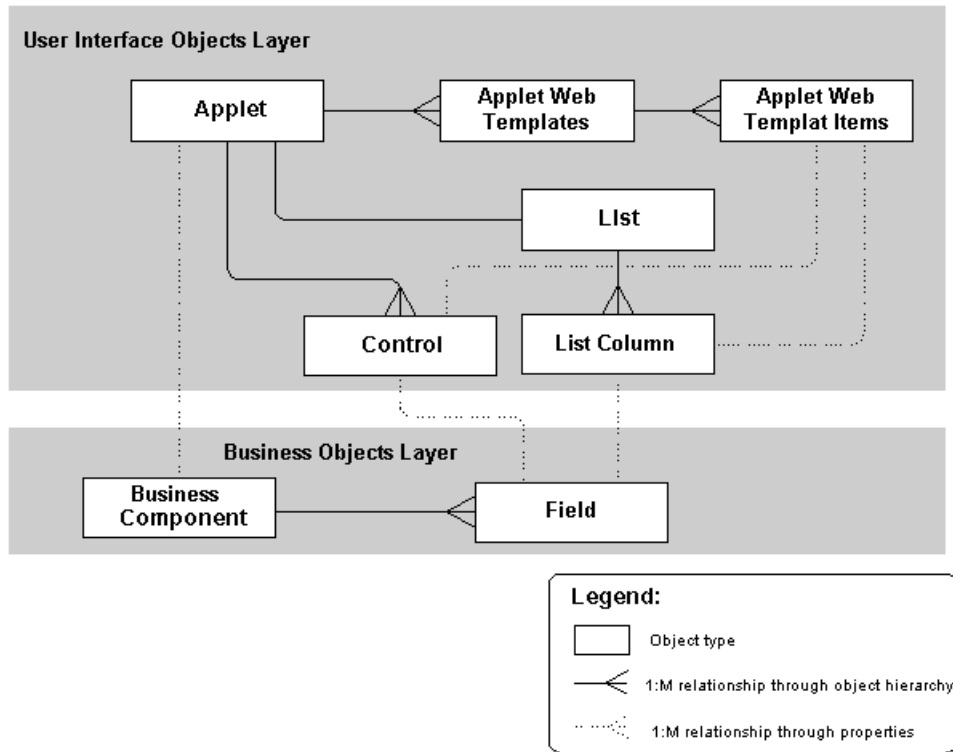


Figure 144. List Applet Architecture

A list applet consists of an applet object definition, a list object definition, multiple list column object definitions, and multiple control object definitions (including, at a minimum, the list control). These object definitions have the following roles in a list applet:

- **Applet object.** The applet object definition provides the properties that apply to the entire applet, such as the Name, Business Component, Width, and Height. It specifies in the Class property that the applet is a list applet (CSSFrameList class).

- **List object.** The list object definition provides properties that govern how the list table operates, such as whether or not totals appear at the bottom of numeric columns. Configuration of the list object definition is described in [“Configuring the List” on page 476](#).
- **List Column object.** Each list column object definition identifies one list column in the scrolling list table. A list column corresponds to one field in the business component. Configuration of the list columns is described in [“Configuring the List Columns” on page 476](#).
- **Control object.** Each Control object definition identifies one visual construct in the list applet. With the exception of the list control, they all appear outside of the scrolling list table. Typical controls in the list applet include the Title control, which indicates the current record and the total number of records listed. Sometimes a set of navigation command buttons are also included in a list applet. Configuration of the controls in the list applet is described in [“Configuring Controls in a List Applet” on page 478](#).
- **Applet Web Template object.** Associates an applet to a Web template. Web templates determine the layout and format of the applet when it is rendered in the user interface. An applet can be displayed in five modes. An Applet Web Template is defined for each mode. The modes are:
 - **Base.** Read only.
 - **Edit.** Used for editing records where users can update values. You can also use the edit for creating new records and querying.

NOTE: Typically, New and Query modes are not necessary because Edit mode can be used for these type of actions.

- **New.** Used for creating a new record where the requirements for new mode are different from the edit mode.
- **Query.** Used for querying where the requirements for the query mode are different from the edit mode.
- **Edit List.** Used for editing records in a list applet.

- **Applet Web Template Items object.** Maps controls to placeholder tags in a Web template. They contain the name of a control or list column as well as unique identifier of a template placeholder. The placeholder determines its position of the control or list column in the Web page rendered at runtime. Applet Web Template Items are automatically populated when users drag and drop controls into placeholders using the Applet Web Layout editor.

For more information about mapping controls and list columns to Web templates, see [“Editing the Web Layout of Applets” on page 544](#).

Configuring the List

List is a child object type of Applet. A list applet has exactly one list object definition, named List. The List object definition provides property values that pertain to the entire scrolling list table, and it serves as a parent object definition for the list column object definitions.

A List object definition (and a list control) are created when you create a new list applet in the Applet wizard. In an existing list applet you can select the list object definition by clicking the list table. You can then edit the properties in the Properties window or use the Object List Editor.

Configuring the List Columns

List Column is a child object type of List. It identifies one column in the scrolling list table and corresponds to one field in the business component. The List Column object type has the following important properties:

- **Name.** The name of the list column, for reference by other object definitions. The Name must be unique among the child list column object definitions of the list.
- **Field.** This property identifies the field from which the list column derives its values.
- **HTML Sequence.** This property defines the tab sequence when the list applet is shown for certain actions like query, new, edit, and so on. It does not apply to list columns.
- **Display Name.** The text which appears at the top of the list column, identifying the column.

- **Display Format.** A format specification for data displayed by the list column. It is used for numeric, date, currency, and similar non-text data types. For details, refer to [“Form Applet Controls” on page 465](#).
- **MVG Applet.** This property identifies which applet to use for the pop-up multi-value group if the field for the list column is a multi-value field.
- **Pick Applet.** This property identifies the applet to use for the pop-up pick applet if the field for the list column has a picklist specified.
- **Text Alignment.** Specifies the alignment of text in the list column. Valid values are Left, Center, and Right.
- **Total Required.** A TRUE/FALSE property indicating whether the list column is to be totaled. An example of a list applet with totals in a list column is the Expense Item List Applet.
- **HTML Type.** Specifies the style of the control. Examples of control types include Field, Text, ComboBox, CheckBox, Button, and Active X. See the section [“Form Applet Controls” on page 465](#) for descriptions of common control types.

For a complete description of properties of the List Column object type, see *Object Types Reference*.

Configuring Controls in a List Applet

The controls in a list applet generally appear outside of the scrolling list table and control the behavior of the scrolling list table or display information about the list table. However, some controls, such as the Label in [Figure 145](#), appear inside the scrolling list table.

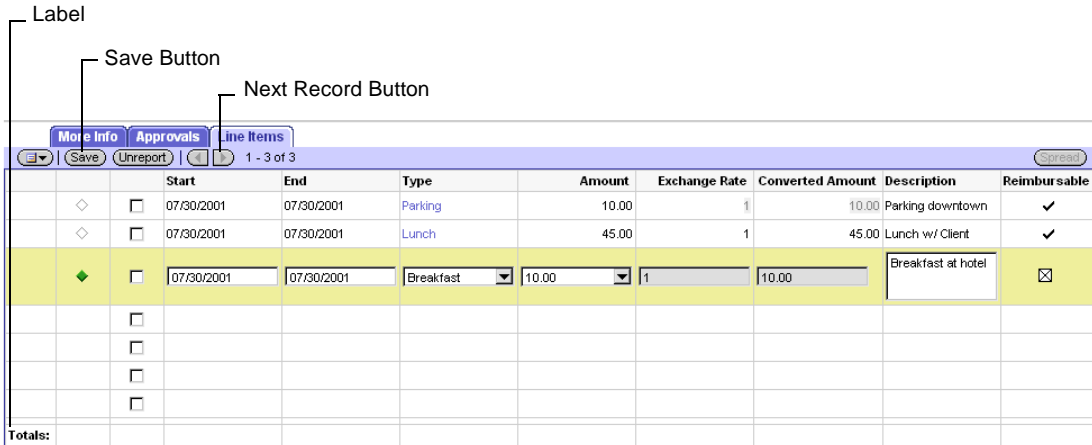


Figure 145. Controls in a List Applet

For more information about controls, see [“Form Applet Controls”](#) on page 465.

About HTML Control Types

HTML control types are described in [Table 34](#).

Custom controls are defined in *SIEBSRV_ROOT\WEBTEMPL\CCHTMLType.swf*, *dCCHTMLType.swf*, and *CfgHTMLType.swf*. The SWE framework references these files for information on how to display the controls (only base controls are defined in SWE). Users can add additional custom types.

Table 34. HTML Control Types

Display Value	Description
ActiveXControl	Allows the placement of an ActiveX control in the applet.
Button	<p>This type can be used with controls that have a Method Invoked property defined. This can be a built-in method supplied with Siebel applications or a custom method programmed in Siebel VB or Siebel eScript. It creates a UI element that when clicked invokes the method.</p> <p>The Runtime property of a button control must be set to TRUE. Otherwise the method associated with it will not execute.</p> <p>To enable a button the WebApplet_PreCanInvokeMethod event must be scripted to set its CanInvoke parameter to TRUE.</p> <p>eScript example:</p> <pre>function WebApplet_PreCanInvokeMethod (MethodName, &CanInvoke) { if(MethodName == "Map") { CanInvoke = "TRUE"; return(CancelOperation); }else { return (ContinueOperation); } }</pre>
ButtonDiv	Custom control, a divider that separates buttons.

Table 34. HTML Control Types

Display Value	Description
Caption	<p>Similar to the Label type. The difference between the Label and Caption type is in the SWE tag syntax used for this control in the Web template.</p> <p>To show a control of type Caption, use the syntax <code><swe:control property="FormattedHtml" /></code>.</p>
CheckBox	Creates a UI element that supports toggling between two states. Check boxes are used for Yes/No or True/False options.
ComboBox	Creates a UI element that allows the selection of a value from a set of values. This type can be used only if the control has a picklist defined that provides the list of values.
Div	Custom control, a divider.
DrilldownTitle	Custom control, a title of an applet that can bring the user to the appropriate view. This is used frequently by applets on the home page.
Field	Custom control, a field label for a list applet.
FieldLabel	Custom control, a field label for a form applet.
File	Creates a UI element that can be used to attach a file.
FormSection	<p>Custom control, a label that helps to group related fields in an applet. The FormSection label expands to fit the region where you place it. To set it apart, the label appears against the FormSection color defined in the cascading style sheet.</p> <p>The control might not appear to expand to fit within the layout editor but it will render that way in production.</p>
Hidden	Creates an HTML input of type Hidden. Such controls are not visible in the Web page but can be accessed through scripting.
ImageButton	Custom control, an image-based minibutton. See the description for MiniButton.

Table 34. HTML Control Types

Display Value	Description
Label	<p>Allows a label to be placed in the applet. A label is a text string that remains constant rather than displaying dynamic information.</p> <p>To show a control of type Label the property attribute can be set to either DisplayName or FormattedHtml. This type was added to handle a special case where the above difference is important. Normally, the type Label should be used instead of Caption.</p>
Link	<p>Used with controls that have a Method Invoked property specified (this could be a built-in method (supplied with Siebel applications). Creates an HTML hyperlink that will invoke the method when activated.</p>
MailTo	<p>Used with controls that contain an email address. The control value will be displayed as a link which when activated will open the user's default email program with the address filled in with the control value.</p>

Table 34. HTML Control Types

Display Value	Description
MiniButton	<p>Custom control that produces a rounded button. MiniButton is the standard button control in Siebel applications.</p> <p>Used with controls that have a Method Invoked property defined. This can be a built-in method supplied with Siebel applications or a custom method programmed in Siebel VB or Siebel eScript. When the button is clicked, the method is invoked.</p> <p>The Runtime property of a button control must be set to TRUE. Otherwise the method associated with it will not execute.</p> <p>To enable a button the WebApplet_PreCanInvokeMethod event must be scripted to set its CanInvoke parameter to TRUE.</p> <p>eScript example:</p> <pre>function WebApplet_PreCanInvokeMethod (MethodName, &CanInvoke) { if(MethodName == "Map") { CanInvoke = "TRUE"; return(CancelOperation); }else { return (ContinueOperation); } }</pre>
MiniButtonEdit	<p>Custom control that displays a button when the applet is in Edit mode. See the description for MiniButton.</p>
MiniButtonEditNew	<p>Custom control that displays a button when the applet is in Edit or New mode. See the description for MiniButton.</p>
MiniButtonNew	<p>Custom control that displays a button when the applet is in New mode. See the description for MiniButton.</p>

Table 34. HTML Control Types

Display Value	Description
MiniButtonQuery	Custom control that displays a button when the applet is in Query mode. See the description for MiniButton.
Password	Creates a UI element that can be used to input a password field. The characters entered in this control will be masked by the * character.
PositionOnRow	Custom control that shows the currently selected record in a list.
RTCEmbedded	Custom control, an embedded text editor.
RTCEmbeddedLinkField	Custom control that allows you to display graphics and links in the RTCEmbedded object.
RadioButton	Can be used in standard interactivity instead of the combo box to show the choices as a radio button.
RecNavNxt	Custom control used to display the next set of records.
RecNavPrv	Custom control used to display the previous set of records.
SSNxt	Custom control used to display the next question in a SmartScript.
SSPrv	Custom control used to display the previous question in a SmartScript.
Text	Creates a UI element that can be used to enter text.
TextArea	Creates a UI element that can be used to enter text in multiple lines.
URL	Used with controls that contain URL values. The value will be displayed as a hyperlink which when activated will take the user to the URL.

About the Display Format Property

You can specify an explicit format mask in the Display Format property of a control (in a form applet) or list column (in a list applet) using various symbols.

For DTYPE_DATETIME you can also specify Date, Time, or TimeNoSec in the Display Format property that will display the specified portion using the format in the Windows Control Panel. The Display Format property is ignored for values of DTYPE_PHONE.

About the Type Property

For controls or list columns based on fields with certain data types, Siebel applications enable certain controls to pop up at runtime, as shown in [Table 35](#).

To cause a runtime pop-up control to appear, the Runtime property of the list column or control must be set to TRUE.

Table 35. Runtime Pop-Up Controls

Field Data Type	Pop-Up Control
DTYPE_DATE	Calendar
DTYPE_TIME	Time
DTYPE_DATETIME	Combination calendar/time
DTYPE_NUMBER	Calculator
DTYPE_INTEGER	Calendar

If there is a picklist defined for a field that has one of the types mentioned above, then a picklist pops up at runtime, instead of a calculator or calendar.

NOTE: The Read Only property of the list column or control must be set to FALSE to use the Runtime property to access runtime pop-up controls.

About the Search Specification Property

If the value in the Search Specification property in an Applet object definition is non-blank, the set of records provided to an applet is restricted. The search specification contains the names of one or more fields in the business component and various operators, combined to create a conditional expression. Records in which the value of the conditional expression evaluates to TRUE are provided to the applet for display; those records in which the expression evaluates to FALSE are excluded.

NOTE: Search specifications on child applets are not executed.

Some sample search specification expressions appear below:

```
[Type]= "COST LIST"
```

```
[Revenue] > 5000
```

```
[Competitor] IS NOT NULL and [Competitor] <> "N"
```

```
[Type] = LookupValue ("TODO_TYPE", "In Store Visit")
```

Search specification expressions are built according to the following syntax rules:

- Standard comparison operators are used to compare a field to a constant, or one field to another field. These include =, < >, >, <, > =, and < =.

Example: [Revenue] > 5000

- String constants are enclosed in double quotation marks. String values are case sensitive, so the use of uppercase and lowercase letters in the search specification should exactly match that of the records you want returned.

Example: [Type] <> "COST LIST"

- The logical operators AND, OR, and NOT are used to negate or combine expressions. Case is ignored in these operators; for example, "and" is the same as "AND").

Example: [Competitor] IS NOT NULL and [Competitor] <> "N"

- A field name in a search specification must be enclosed in square brackets.

Example: `[Conflict Id] = 0`

- The LIKE operator can be used to create text string comparison expressions in which a field is compared to a constant, or a field to another field, and a match on only the first several characters is required. The wildcard characters “*” and “?” are used to indicate any number of characters, and a single character, respectively.

Example: `[Last Name] LIKE "Sm*"`

In this example, the Last Name values of Smith, Smythe, Smallman, and so on would cause the expression to evaluate to TRUE.

- The search specification expression must be 255 characters or less.

An applet search specification cannot be used to override the search specification of the underlying business component, if the business component has one. Rather than overriding the business component’s search specification, the applet’s search specification is appended to that of the business component. Search specifications should appear in the business component or the applets that use it, but not both.

The search specification on an applet is converted to a WHERE clause by the data manager at runtime. When two applets based on the same business component appear in the same view, one query is generated against the database to populate both applets. Because a database select statement only supports one WHERE clause, only one of the applets should have a search specification—or if both do, they should have the same specification.

For example, the Account List Applet and the Account Entry Applet both appear in the Account List View. The record that is selected in the Account List Applet also appears in the Account Entry Applet. When you select a different row in the list or scroll through the list, the Account Entry Applet is updated to show the same record that is selected in the Account List Applet. This is made possible by the fact that both applets are populated from the same query and therefore show the same record set.

To prevent the two applets from being synchronized, they would have to be on separate business components, for example by copying the business component on which the first applet is based.

For more information on the usage of the Search Specification property of applets, see *Object Types Reference*.

When the Applet Visibility Type property of the View Web Template Item object is set to a non-null value, it might cause search specifications on the applets in that view to be ignored. This property is recommended for use mainly where the applets in a view are based on different business components. If you use this property, test it thoroughly for functionality.

Search specifications can impact performance negatively, particularly when you include fields based on joins in the search specification. Search specifications with NOT or OR can also adversely affect performance by forcing the database to execute a full table scan.

Views

A *view* is a collection of applets that appear at the same time on the same screen. A view can be thought of as a single window's worth of data forms (applets). Generally, a Siebel application window displays one view at any one time. The currently active view is changed by selecting a different view from the view tabs or from a menu suboption in the Site Map.

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 neither recommended nor supported.

Views are typically of the following styles:

- **List-form view.** In a list-form view, a list applet and a form applet display data from the same business component. The list applet appears above the form applet. The form applet presents the same information as the currently selected record in the list applet, with a different arrangement that may include more fields.

List-form views are described in [“List-Form Views” on page 489](#).

- Master-detail view.** In a master-detail view, a form applet and a list applet display data from two business components related by a link. The form applet appears above the list applet. The form applet displays one record from the master business component in the master-detail relationship. The list applet displays all of the records from the detail business component that have as their master record the record currently displayed in the form applet.

Master-detail views are discussed in [“Master-Detail Views” on page 490](#).

NOTE: Master-detail views can display multiple master-detail relationships, with a different applet for display of records from each detail business component.

List-Form Views

In a *list-form view*, a list applet and a form applet display data from the same business component. The list applet appears above the form applet. The form applet presents the same information as the currently selected record in the list applet.

Figure 146 illustrates the Accounts List View, which is a list-form view in Edit mode.

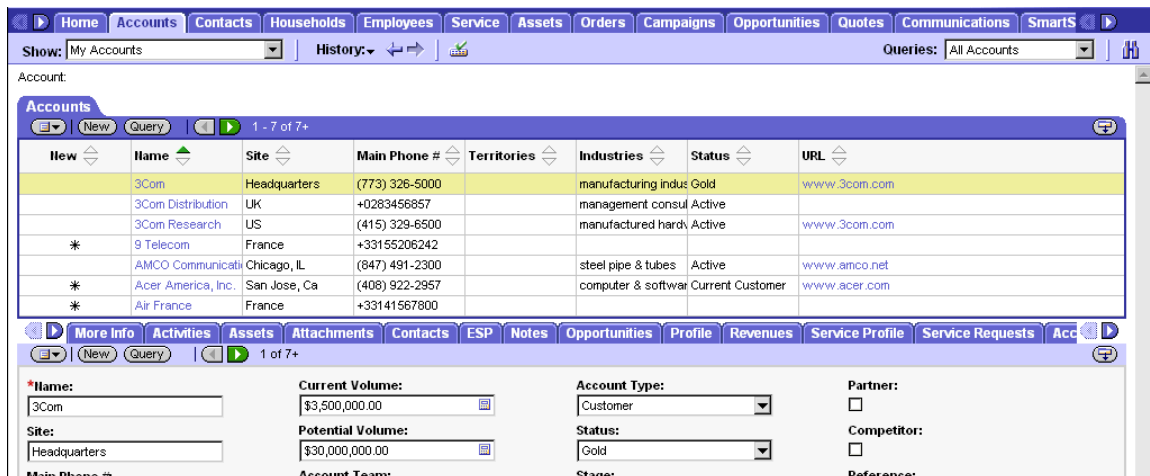


Figure 146. List-Form View

The form applet displays information from the same account, but in a form which can be viewed without scrolling. Notice the Select button and down-arrow icons to the right of some text boxes in the form applet. These indicate that multi-value group applets and picklists are also available from the list applet. Select buttons and down-arrow icons are not visible in the list applet until list column cells containing them are clicked.

The applets in this view are Account List applet and Account Entry applet. Both are based on the Account business component. The Account List view uses the property settings in [Table 36](#).

Table 36. Property Settings in Account List View

Property	Value
Business Object	Account
Title	My Accounts

Master-Detail Views

In a *master-detail view*, typically a form applet and a list applet display data from two business components related by a link. The form applet appears above the list applet. The form applet displays one record from the master business component in the master-detail relationship. The list applet displays all of the records from the detail business component that have as their master record the record currently displayed in the form applet.

NOTE: In another variant of the master-detail view style, the view can consist of two list applets. The records in the detail list applet are detail records of the currently selected record in the master list applet.

Figure 147 illustrates the Contact Detail - Accounts view, which is a master-detail view.

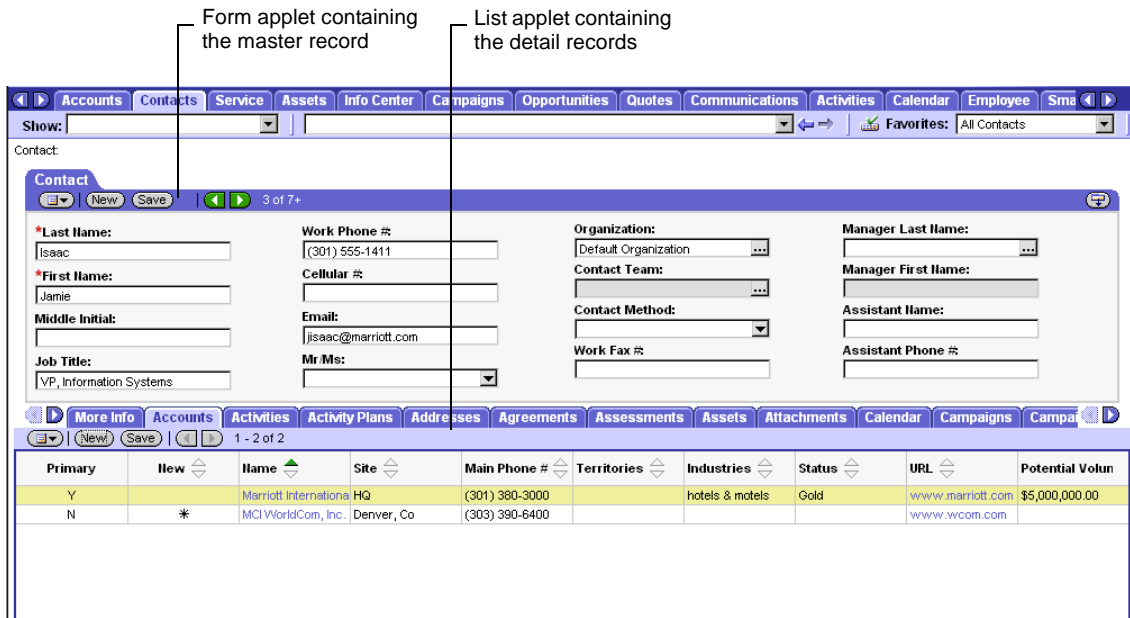


Figure 147. Master-Detail View in a Siebel Application

The list of Accounts for this Contact appears in the list applet. If a different Contact appeared in the form applet, a different set of Accounts would appear in the list applet.

The applets in this view are Contact Form applet and Contact Account applet. They are based on the Contact and Account business components, respectively. The business object associated with the view is Contact. In the context of the Contact business object, the master-detail relationship between Contact and Account is based on the Contact/Contact Account link. See [Figure 148](#) for the relationships among the objects.

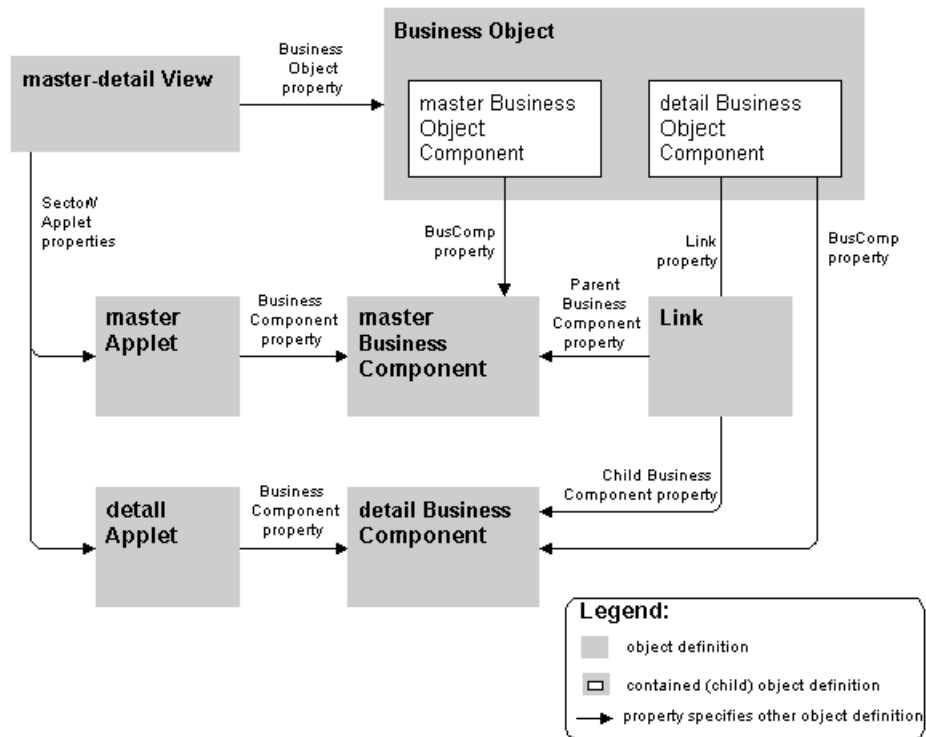


Figure 148. Master-Detail View Architecture

The object definitions in this diagram are briefly described in the following:

- **Master-detail view.** The view being implemented.
- **Master applet.** The form applet used to display the master record.
- **Detail applet.** The list applet used to display the corresponding detail records.

- **Business object.** Business object associated with the view by means of the Business Object property of the View object. The business object establishes the context that determines the active link between the business components associated to the two applets.
- **Business object components.** The business object components are child objects of the business object. Each business object component associates a business component to the business object.
- **Master business component.** The business component associated with the master applet.
- **Detail business component.** The business component associated with the detail applet.
- **Link.** The link that specifies the master-detail relationship between the master and detail business components. It is identified in the Link property of the detail Business Object Component object.

Thread Bars

The *thread bar* is a navigational tool for the user. It provides the means to navigate from view to view among the views previously visited in the current screen. The Thread Bar is identified in [Figure 149](#).

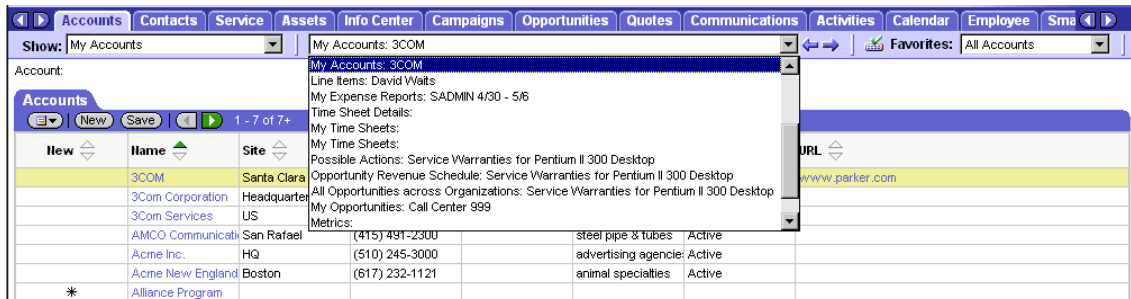


Figure 149. Thread Bar in a Siebel Application

The entries in the thread bar identify a view (based on a different business object) that the user has visited. The name of the view is given in the box.

The following properties in each view object definition are set in order to configure thread behavior:

- **Thread Applet.** Specifies which of the applets appearing in the view supplies the data value for the thread field.
- **Thread Field.** The name of the field whose data value is included in the arrow box, following the Thread Title. This is a field in the business component associated with the applet identified in the Thread Applet property.
- **Thread Title.** The text used in the thread to identify the view. For example, in most of the views displaying Accounts (such as Account List view and Account Detail - Contacts view), the Thread Title is Acct.

Drilldown Behavior in a View

The *Drilldown Object* object type is a child of Applet, used primarily in list applets. It allows the user to drill down from a cell in a list applet (or using a pop-up menu in either a form or list applet) to a particular view. Drilldown controls or list columns in a list applet in Siebel applications consist of colored, underlined text, much like a hypertext link in a Web browser. Drilldowns in a list applet are illustrated in [Figure 150](#).

Drilldowns in Account list column

New	Name	Site	Main Phone #	Territories	Industries	Status	URL
	<u>3COM</u>	Santa Clara	(408) 326-5000		manufacturing industries	Gold	www.parker.com
	<u>3Com Corporation</u>	Headquarters	(415) 329-8500		manufacturing industries	Active	
	<u>3Com Services</u>	US	(415) 329-8500		manufactured hardware (general)	Active	
	<u>AMCO Communications</u>	San Rafael	(415) 491-2300		steel pipe & tubes	Active	
	<u>Acme Inc.</u>	HQ	(510) 245-3000		advertising agencies	Active	
	<u>Acme New England Division</u>	Boston	(617) 232-1121		animal specialties	Active	
*	<u>Alliance Program</u>						

Figure 150. Drilldown List Columns in a List-Form View

NOTE: Drilldown behavior is not supported on MVG applets, pick applets, or association applets.

In the standard (or *static*) drilldown configuration, a specific view is referenced for each hyperlink list column or control. Clicking the hypertext in the list column or control takes you to that view under all circumstances. Another kind of drilldown configuration, *dynamic drilldown*, is discussed in [“Dynamic Drilldown Behavior” on page 496](#).

If the driving applet of a view has a search specification, this search specification is also applied to the destination view when drilling down.

For more information, see [“About the Search Specification Property” on page 486](#) and *Object Types Reference*.

NOTE: If the target view of a drilldown object has a different visibility type from the origin view, drilling down on a cell will take the user to the first record of the destination view and not to the drilldown record.

Static Drilldown Behavior

In the example, underlined account name appears in the list column labeled Name. If the user clicks the account in the Name list column, a master-detail view appears, with the selected account in a form applet above an applet displaying the corresponding list of contacts.

Drilldowns appear as hypertext only in list applets.

Figure 151 displays the property relationships between the list applet, business component, and view in a static drilldown configuration.

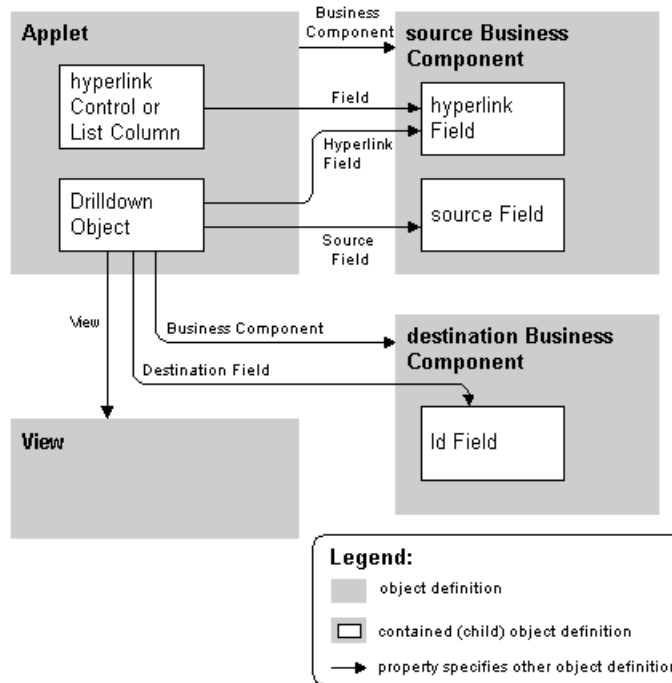


Figure 151. Static Drilldown Configuration

One Drilldown Object object definition is specified for each list column in the list applet to have drilldown functionality.

Dynamic Drilldown Behavior

Dynamic drilldown enables hyperlink navigation to multiple views from the same hyperlink field, depending on the value of a field in the applet's current record.

This is useful in the situation where special processing is desired for various types of contacts, opportunities, accounts, and so on. The business component may have a field that indicates a classification, such as the Lead Quality for an opportunity or the primary Industry for an account. The drilldown behavior can be to check this field in the current record, and navigate to different views for different values found there.

Dynamic drilldown behavior for a hyperlink field (and the corresponding list column or control) is configured with one or more Dynamic Drilldown Destination child object definitions of the Drilldown Object. This is illustrated in [Figure 152](#).

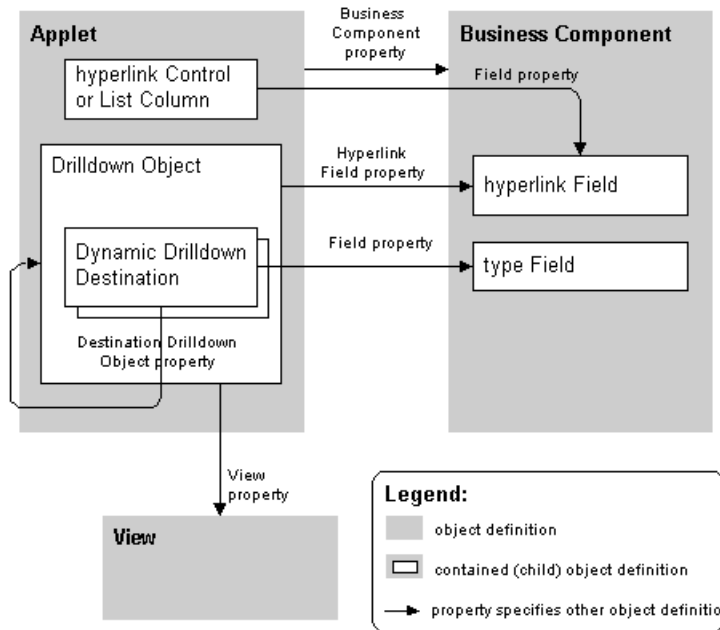


Figure 152. Dynamic Drilldown Configuration Details

As in a static drilldown configuration, the Drilldown Object object definition identifies a hyperlink field and a view. These property settings continue to have the same purpose in dynamic drilldown, namely to specify the list column or control that has hyperlink capabilities, and the destination view when the hyperlink is clicked.

However, for dynamic drilldowns, you define Drilldown objects for each candidate view. The Drilldown object with the *lowest* sequence number contains child Dynamic Drilldown Destination objects that are used to define the conditions under which each of the Drilldown objects should be activated. When the conditions defined in a Dynamic Drilldown Destination are matched, the logic routes to one of the candidate Drilldown objects. When all conditions expressed in the Dynamic Drilldown Destinations are false, the parent Drilldown object acts as the default.

For example, the Industry field in the Account business component could be designated as the type field in a list of Dynamic Drilldown Destinations. When the Industry value is “Manufacturing,” the drilldown could route to a Drilldown Object with a view tailored for manufacturing accounts. When the value is “Transportation,” the destination could be a different Drilldown Object and view, and so on.

The list of Dynamic Drilldown Destinations contained in a Drilldown Object specifies a set of criteria, of which any number may be met. If the condition in one Dynamic Drilldown Destination is met, the hyperlink routes to the specified Drilldown Object. If more than one is met, the first encountered (as specified in the Sequence property) specifies the destination Drilldown Object. If none is met (or no Dynamic Drilldown Destination object definitions are supplied as children of the Drilldown Object), the Drilldown Object itself supplies the name of the destination view.

Be careful to avoid routing hyperlinks from one dynamically evaluated Drilldown Object to another. That is, if you create Dynamic Drilldown Destination children of a Drilldown Object, do not have them route to a Drilldown Object that itself has Dynamic Drilldown Destination children. This practice could lead to ambiguity or looping.

If multiple drilldown objects for the applet are defined, a given field in the business component should be mentioned only once for all available drilldown objects. For a dynamic drilldown, the drilldown object that contains the dynamic drilldown destinations should have the Hyperlink Field property set.

Applet Toggles

Applet toggles allow users to switch back and forth between different applets within the same view. This is useful when you want to display different types of data or present the same data in a different way. There are two types of applet toggles, static and dynamic:

- **Static applet toggles.** Allow users to toggle between applets by selecting the name of the applet from the Show drop-down list.
- **Dynamic applet toggles.** Automatically toggles between applets based on the value of a field in a parent applet.

When configuring applet toggles, consider the following:

- For applets involved in a toggle cycle the search spec on the form applet in the view will be applied first. Therefore, to apply a search spec on a list applet in a toggle cycle, you also need to add the search spec for the form applet.
- Dynamic toggle applets must be based on the same business component.
- Static toggle applets do not have to be based on the same business component.
- You cannot configure more than one applet toggle in a view.

To understand applet toggles consider an example. Suppose you need to store information about your customer's preferred payment method using the contact business component. Payment methods are cash, credit card or check. For each payment method, you need to gather different data:

- **Cash.** No special data requirement, this is the default payment method
- **Credit Card.** Credit Card Type, Credit Card Number, Expiration Date
- **Check.** Checking Account Number, Routing Number, Driver License Number, Driver License State

You could configure this scenario using a static toggle applet or a dynamic toggle applet. A static toggle applet would require the user to toggle between the different applets by selecting it from the Show drop-down list. Dynamic toggle would automatically toggle between applets based on the value entered in a Payment Type field.

To configure applet toggles (an example)

- 1** Create new fields in the Contact business component to capture the following information:
 - Payment Method (use a static bound picklist that contains Cash, Credit Card, and Check)
 - Credit Card Type
 - Credit Card Number
 - Expiry Date
 - Checking Account Number
 - Routing Number
 - Driver License Number
 - Driver License State
- 2** Expose the Payment Method Field in the Contact Form Applet. This is the default applet that would be used when the contact's preferred payment method is Cash.
- 3** Create two copies of the Contact Form Applet.
Name one of them Contact Form Applet - Credit Card and the other Contact Form Applet - Check.
- 4** Expose the following fields in the Contact Form Applet - Credit Card:
 - Credit Card Type
 - Credit Card Number
 - Expiry Date BC Fields

NOTE: When the contact's preferred payment method is Credit Card, the Contact Form Applet - Credit Card applet allows the user to enter a contact's credit card information.

- 5** Expose the following fields in the Contact Form Applet - Check:

- Checking Account Number
- Routing Number
- Driver License Number
- Driver License
- State BC Fields

NOTE: When the contact's preferred payment method is Check, the Contact Form Applet - Check applet allows the user to enter a contact's checking account information.

- 6 Find and select the Contact Form Applet in the Object List Editor, navigate to the Applet Toggle child object, type and create the following records:
 - **Record 1:**
 - Applet = Contact Form Applet - Check
 - Auto Toggle Field = Payment Method
 - Auto Toggle Value = Check
 - Name = Contact Form Applet - Check
 - Parent Name = Contact Form Applet
 - **Record 2:**
 - Applet = Contact Form Applet - Credit Card
 - Auto Toggle Field = Payment Method
 - Auto Toggle Value = Credit Card
 - Name = Contact Form Applet - Credit Card

- Parent Name = Contact Form Applet

NOTE: To configure this example as a static toggle applet that allows the user to select the appropriate applet from the Show drop-down list, leave the Auto Toggle Field and Auto Toggle Value properties blank.

- 7 Compile your changes and test.

Screens

A *screen* is a logical collection of views. It is not a visual construct in itself; rather, it associates views so that other visual constructs, such as the Site Map and tab bar, can reflect the list of views contained in the currently active screen.

A screen does not have a direct relationship with a business object in the same way that a view does. No property in the Screen object type specifies a business object. However, a screen normally contains only views relating to the same business object; this is good design practice. In this sense, it can be loosely said that a screen corresponds to one business object.

You can select a screen from the Site Map or from the first level of tabs shown in Figure 153.

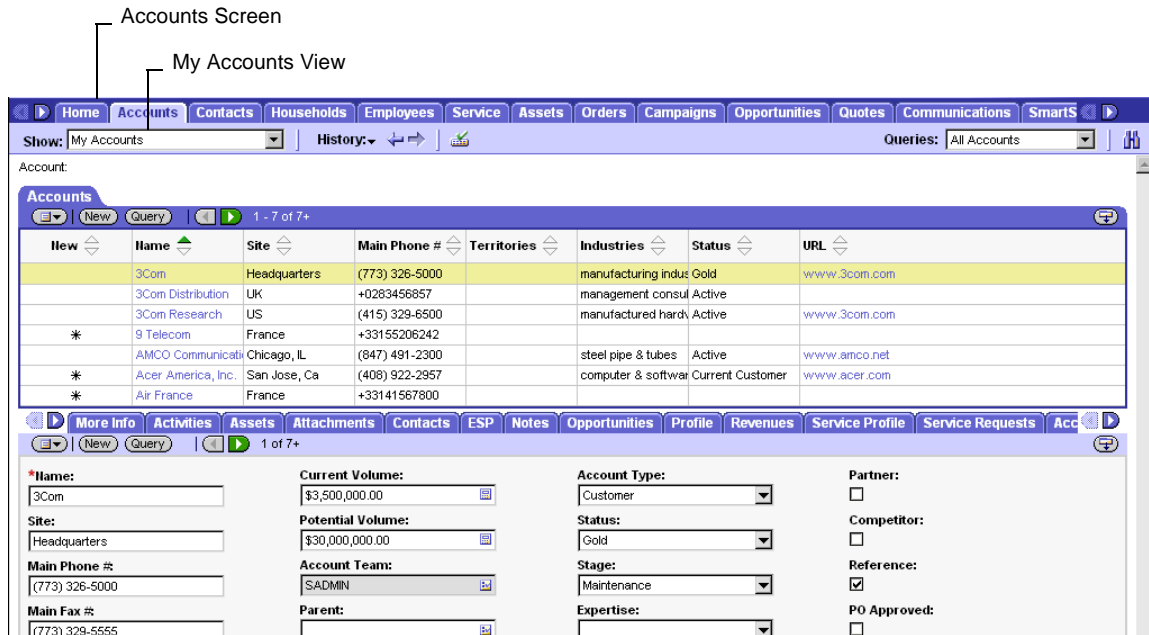


Figure 153. First Level Navigation Tabs

A screen is implemented as a Screen object definition with Screen View child object definitions. Each Screen View object definition associates an existing view with the screen. Properties within each screen view object definition specify the screen's appearance in the view tabs and Site Map.

NOTE: You can create two different views based on different business components; however, in that situation the Siebel application cannot keep track of record context and you will not be able to navigate between them.

About The User Interface Navigational Paradigm

The user interface contains four levels of navigation.

- 1** Top level tabs (Page Tabs) allow users to navigate between screens.
- 2** Views displayed in the *Show: combo box* on the toolbar allow users to navigate between views. These views are often context views controlled by visibility properties, such as My Contacts, All Contacts, and My Team's Contacts. They can also be a list of all available views in a screen, such as the views displayed in the Application Administration screen.
- 3** View tabs (displayed between the top applet and the bottom applet in a typical view) allow users to navigate between master-detail views to see different kinds of data.

- 4 Views displayed in a *Show: combo box* that appears in the banner of a subview allow users to select between different subviews, such as different chart views as shown in [Figure 154](#).

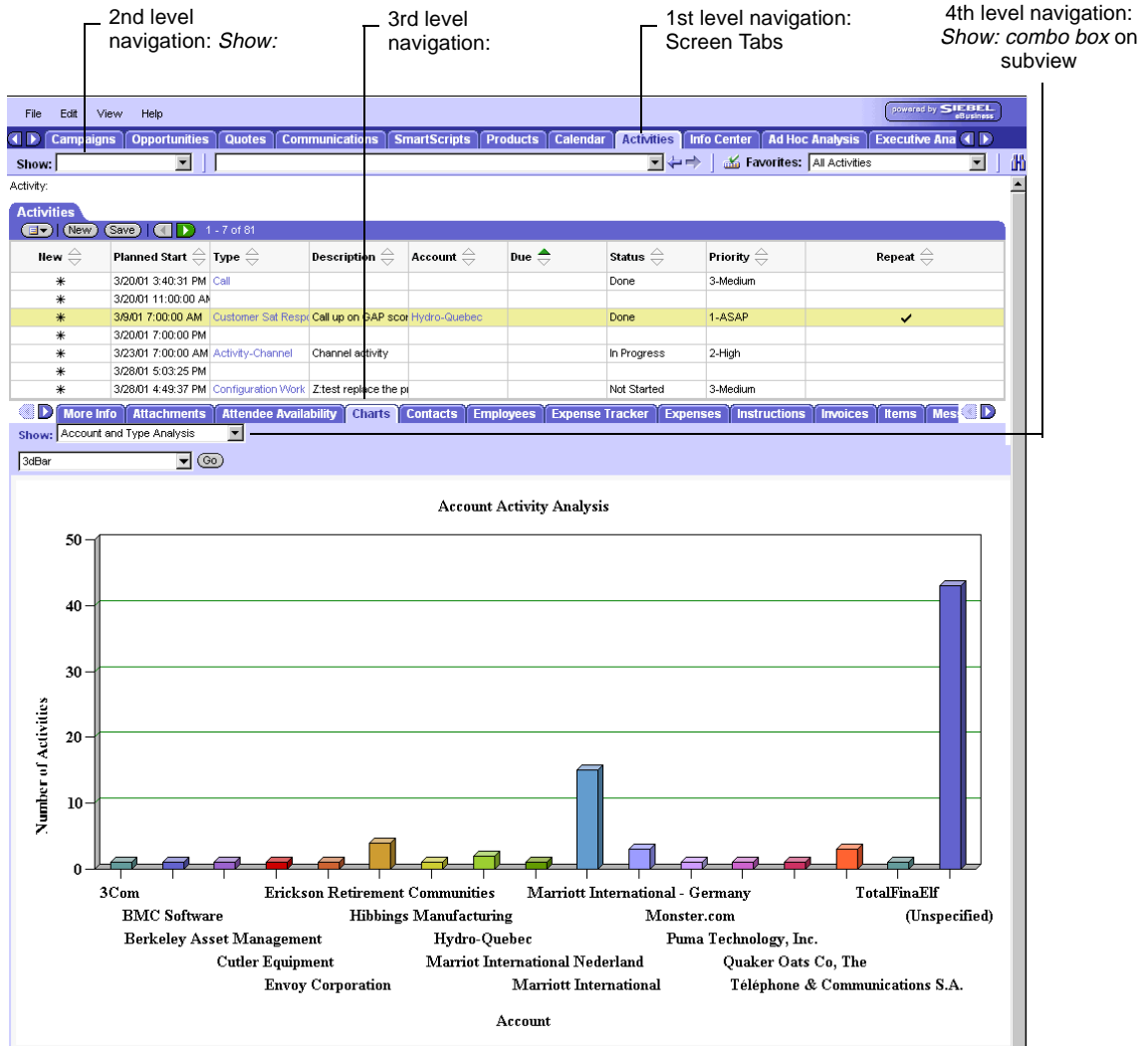


Figure 154. Four Levels of Navigation

Each of these navigation levels are controlled by the following objects:

- **Page tabs.** Child objects of the Application object type. Page tabs appear as the first-level navigation tabs in the user interface. Each page tab is associated with a screen. Users select the tab to navigate to a particular screen.

See [“Associating Screens with Page Tabs” on page 577](#) for more information.

- **Views.** The Visibility Applet property of the view object determines whether a view appears in the *Show: combo box* on the toolbar or as a View tab.

If this property is not null, then the view appears in the *Show: combo box*. For example, views such as My Accounts, All Accounts, and My Team's Accounts appear in the *Show: combo box* because the visibility property is set to a non-null value.

However, if the Visibility Applet property is set to a null value, then the view appears as a View tab, as is typically the case with master-detail views.

It is useful to group views this way, because it allows you to separate the views that show the same type of data, but filter it differently based on visibility rules, from views that show different types of data. Views that show the same data but filter it differently appear in the *Show: combo box* and views that show different types of data appear as View tabs.

See [“About Views” on page 562](#) for more information.

When a query is performed in a view or a predefined query is executed for a view, the current query context remains when users navigate to another view using a view tab, where the driving applet is based on the same business component as the previous view.

However, a fresh query is applied when the view is visited from the Site Map.

The visibility rules of the new view are also applied. If a query is entered in an All view and then a View tab is used to navigate to a Manager view or a Sales Rep view, only the subset of records that is visible to the user as a manager or as a sales representative will be seen.

If there is a search specification on a driving applet in a view and the user navigates from this view to another using a view tab, the search specification is applied to the destination view—keeping the same query.

An explicit search specification can be specified for the applet in the target view to force a new query. For more information, see [“About the Search Specification Property” on page 486](#) and *Object Types Reference*.

- **Screen view.** A view appears in the *Show: combo box* on a subview (fourth-level navigation) if the Screen View with which the view is associated has non-null values for the following properties:

- **Category.** This is a logical representation of what category the view belongs to.
- **Category Menu Text.** The display value that appears within the combo box.
- **Category Viewbar text properties.** The display text of the View tab under which it appears.

NOTE: The values for Category and Category Viewbar text are generally the same.

See [“About Screens” on page 572](#) for more information about screen views.

Applications

Siebel applications are primarily a collection of screens that users can invoke from the desktop by double-clicking an icon or by pointing a browser to a server running the application. Each combination of screens that is appropriate to a specific class of users can be provided as an application. Siebel Sales, Siebel Service, and Siebel eMarketing are examples of applications. Custom applications can be configured as well, uniquely combining user interface object definitions to meet particular requirements of the organization. However, developers should rarely need to do this.

In addition to collecting a group of screens and their views, an application object definition includes the following:

- Find object definitions that configure the Find dialog box.

For more information, refer to [“Screens” on page 502](#).

- Scripts written in Siebel VB or Siebel eScript and browser JavaScript that can be implemented as event procedures on startup, prior to closing, and so on. These are implemented through Application Script child object definitions, and created and maintained in the Siebel VB or Siebel eScript Editor.

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

- Custom menu options for Siebel-provided methods. These are implemented through the application method menu item object definitions, and created in the Applet Method Menu Item Wizard.

See [“Defining Web Menus Using the Applet Method Menu Wizard” on page 591](#) for more information.

NOTE: Applets can have their own custom menus as well.

A desktop icon is configured to activate a specific application through the `/c` command line switch and the `ApplicationName` parameter in the configuration file.

Screens are included in an application object definition using page tab and screen menu item child object definitions. Each page tab or screen menu item object definition associates a screen to the application. The page tabs add screens to the Tab bar. The screen menu items add screens to the Site Map. Note that the list of screens in the Tab bar can be different from the list of screens in the Site Map. Typically the site map will be the more complete of the two if the Tab bar and Site Map differ. The set of screens in an application is the union of the screens found in the Tab bar and Site Map.

Web-Related Objects

This section provides a general description of the Web-related objects in Siebel Tools.

- **Web Page.** Defines attributes that are mapped to a Web page template. Web pages contain views and other persistent objects. The relationship between a Web page and a view is established using tags within the Web Page SWT template. Web Page objects/templates are also used for displaying Login, Error pages, and so on.
- **Web Page Item.** Item that can be shown on a Web page. Similar to control, except it is not placed on an applet.
- **Web Page Item Parameter.** Parameter of the Web Page Item. Definition varies depending on the item. For example, for a page item that has `Invoked Method` set to `GoToView`, an optional parameter is `view`, and its value is the name of the destination view.

- **Web Template.** Establishes a relationship between a representation of an SWT file in the repository and the actual file stored in a file directory.
- **View Web Template.** Establishes an association between a parent View object and a Web Template. Each View object can have at most one View Web Template child.
- **View Web Template Item.** Siebel Applets mapped to View Template.
- **Applet Web Template.** Establishes an association between a parent Applet and a Web Template. There are four types: Base, Edit, New, and Query, corresponding to the record operation performed within each. Applets may have several Edit and New Applet Web Templates but will have at most one Base and Query Applet Web Template.
- **Applet Web Template Item.** List items, applet controls, and Web controls that belong to Applet Web Template.
- **Applet Server Script.** Script associated with an Applet that is exposed in an application (through one or more Applet Web Templates). Web scripts can be used to modify output from the Siebel Web Engine (SWE) before it is sent to the browser.

Table 37 describes the new Web-related properties of existing Siebel objects.

Table 37. Existing Siebel Objects with New Web-Related Properties

Web-related Control Attributes	Description
HTML Display Mode	<p>Values are:</p> <ul style="list-style-type: none"> ■ DontEncodeData. Leaves the data as is. This is useful if the configurators wish to put their own HTML in the Caption property and expose the caption to the Web page. ■ EncodeData. Let Siebel Web Engine convert data into HTML format. ■ FormatData. Replace the data. By default, <CR> is replaced with
.
HTML Row Sensitive	<p>Used to indicate to the SWE that it must reposition the Applet on the correct row before invoking the method defined in the MethodInvoked property. Within applets there are controls which can invoke methods. Some of these methods require the existence of a current row before the method can be invoked. For example, DeleteRecord, EditRecord require a specific record, and NewRecord, ExecuteQuery do not.</p> <p>If you set this to TRUE, SWE will try to locate the record first before invoking the method. If the method for your control does not need the current row, you must set it to FALSE. If set incorrectly, SWE may return an error message.</p> <p>For these methods HTML Row Sensitive should be set to TRUE: CopyRecord, DeleteRecord, Drilldown, EditField, EditRecord, PickRecord, UndoRecord, WriteRecord, and GetImage.</p> <p>For these methods HTML Row Sensitive should be set to FALSE: CloseApplet, AddRecord, CreateRecord, ExecuteQuery, ExecuteSort, Find, GotoFirstSet, GotoLastSet, GotoNextSet, GotoPreviousSet, NewQuery, NewRecord, NewSort, ResetRecord, SortAscending, SortDescending, and SortOrder.</p>
HTML Type	<p>The type to be used in the Web client. Values are Button, CheckBox, ComboBox, File, Hidden, Label, Link, Mailto, Password, Text, TextArea, and URL. Used by the Web engine only when the corresponding placeholder tag appears within a form tag.</p> <p>For information on HTML control types, see Table 34 on page 480.</p>
Method Invoked	<p>Siebel Method invoked by the control (frequently used for buttons). Some methods require a Control or List Column User property to specify an argument. For example, the GotoView method requires that the destination view be specified in a User Property.</p>

Table 37. Existing Siebel Objects with New Web-Related Properties

Web-related Control Attributes	Description
Field	Field which will be used to retrieve data from the business component.
Caption	Used with corresponding tags to display the Caption value of the control or list column. If necessary, the Caption property can be used to store unformatted HTML tags. For example a help button might have the following value for its Caption property: <code>Help</code>

Search and Find Objects

Search objects and *Find objects* can be configured to meet your organizations requirements.

- **Find.** Allows users to query the database on a field-by-field basis from anywhere in a Siebel application. Find objects define what objects and fields to query against.
- **Search.** Allows users to perform full text searches across multiple business components and files with one operation. Search objects are a logical entities that define all search characteristics and behaviors for an object. They include attributes that define the business component and fields to search upon, drill-down results views, and records which may be associated with a particular result.

For detailed information about Search and Find objects and related configuration information, see *Siebel Search Administration Guide*.

Toolbars and Menus

Toolbars and *menus* allow users to initiate various actions. The application-level menu (File, View and Help) appears in its own frame near the top of the application in the browser window, and the toolbar appears just beneath the primary tab bar, as shown in [Figure 155](#).



Figure 155. Toolbar and Application-Level Menu

The applet-level menus are invoked from the applet menu button, in the banner at the top of an applet. See [Figure 156](#).

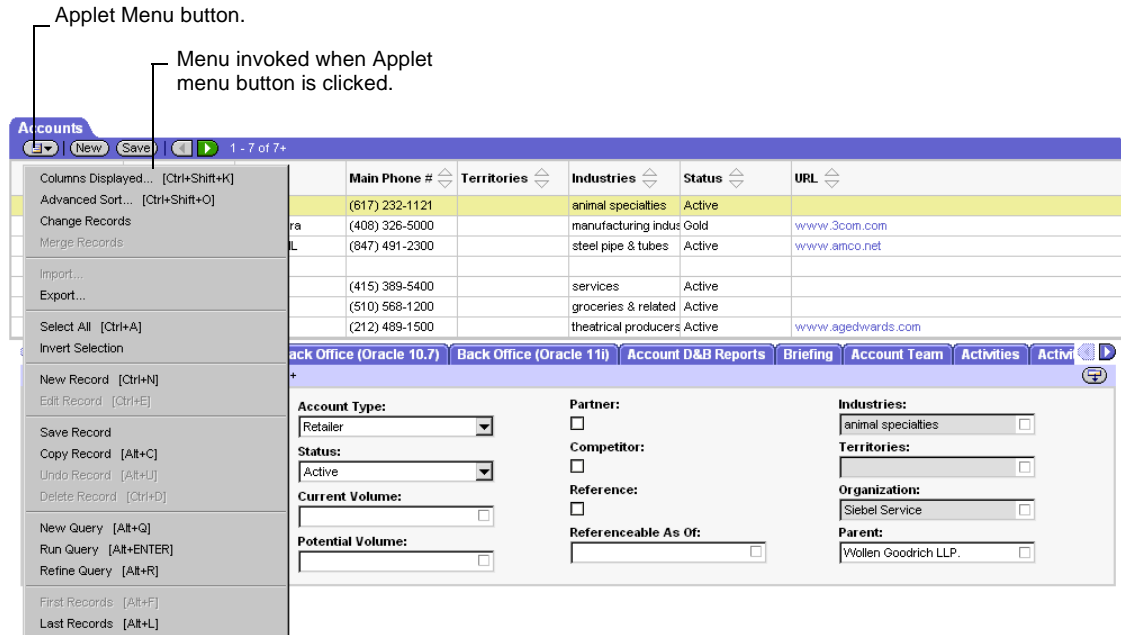


Figure 156. Applet Menu

The user's click on a toolbar icon or menu item is normally translated into a call to an invoke method, which may reside in a service on the browser or server, or in classes in the browser application or server infrastructure (applet or business component classes, SWE frame manager, or model). The toolbar icon or menu item is configured to target a method name, a method handler (from which it may be automatically retargeted if not found), and optionally a service.

Application-level items (which include both toolbar icons and application-level menus) are implemented through the use of Command object definitions in Tools, which are then mapped to Toolbar Item or Menu Item object definitions.

In SWE templates, the `<swe:toolbar>` tag specifies a named toolbar (where the name corresponds to the Name property in the Toolbar object definition in the repository), and the `<swe:toolbaritem>` tag between the toolbar start and end tags recursively retrieves all of the toolbar items for that toolbar from the repository.

See [“Toolbar Template Configuration” on page 866](#) for more information about Toolbars and `swe:` tags.

Toolbar and Menu-Related Object Types

The relevant object types for configuration of menus and toolbars in Tools are described in the following sections.

Command Object Type

A Command object definition specifies which invoke method is called when a toolbar icon or application-level menu item associated with the command is executed (applet-level menus do not use command objects). It also specifies which bitmap appears on the toolbar icon for toolbar items. Command object definitions are referenced by Toolbar Item or Menu Item object definitions.

See [“Using the Command Object Wizard” on page 586](#) for more information about creating Command objects.

A Command object definition has the following significant properties:

- **Target.** Specifies which entity handles the invoke method the command calls. Available options are the following:
 - **Browser.** The method handler is a JavaScript service on the browser, or the JavaScript application, depending on whether a service is specified in the Business Service property.
 - **Server.** The method handler is an object manager service on the server or the object manager infrastructure (the SWE UDF loader, or, secondarily, the model), depending on whether a service is specified in the Business Service property.
 - **Browser Applet.** Used with high interactivity.

For more details on the configuration of the Target property and related properties, see [“Invoke Method Targeting” on page 521](#).

- **Business Service.** Specifies the service (either browser or server, depending on the Target property) that handles the invoke method. If the property is left blank, the browser or server infrastructure is targeted rather than a specific service. If a service is specified, it must handle CanInvokeMethod and InvokeMethod for the method specified in the Method property.
- **Method.** Specifies the name of the method to invoke when the menu item or toolbar icon is selected. This is a required property.

See [“Invoke Method Targeting” on page 521](#).
- **Method Argument.** Provides the means to pass an argument to the invoke method specified in the Method property. For example, a command item that opens a new window and navigates to a URL in that window can specify the GotoURL method in Method and the URL to navigate to in Method Argument.
- **Show Popup.** If TRUE, specifies that a new browser window is opened before invoking the method. If FALSE, specifies that the method is invoked in the current browser window.
- **HTML Popup Dimensions.** Dimensions, in pixels, of the pop-up window, when Show Popup is TRUE. An example is 640x480 (specified with the “x” and without blank spaces).
- **HTML Bitmap.** Specifies bitmap used by the Command object.
- **Tooltip Text.** This is the tooltip text which appears when the cursor lingers on a toolbar icon. For built-in methods, the tooltip text should be left blank; blank indicates that the method will dynamically supply the text, and language localization takes place as a part of this process. For developer-defined methods, you should enter literal text (but note that this turns off language localization for this tooltip text).

Toolbar Object Type

For each toolbar in the application, you create a Toolbar object definition in the Object List Editor. This provides a named toolbar that the user can activate or deactivate in Siebel applications, and to which icons (Toolbar Item object definitions) can be associated or removed. Typical toolbar functionality for most applications is implemented using HTML toolbars. In an HTML toolbar, the buttons are static images, which may be dimmed to indicate unavailability, but otherwise are not manipulated by program logic on the browser. In contrast, communications toolbars in applications such as Call Center, require toolbar icons that can be altered in response to events, such as blinking a particular toolbar icon when a call is incoming. This requires the use of Java toolbars. To specify that a toolbar is of the Java type, a class name is entered in the Class property.

For more information about configuring communications toolbars, see *Siebel Communications Server Administration Guide*.

Important properties of Toolbar are:

- **Class.** Left blank for an HTML toolbar, specified for a Java toolbar - the name of the Java class that implements the toolbar is entered.
- **Name.** Referenced by other object definitions, and by the `<swe:toolbar>` tag in the "name =" clause.
- **Display Name.** Used for the History button and to show or hide toolbars by name.

Toolbar Item Object Type

The Toolbar Item object type associates a Command object definition (identified by name as a property in the Command property) with a Toolbar object definition (the parent of the Toolbar Item). This association places a toolbar icon, whose bitmap image, invoke method, and target are specified in the Command object definition, on the specified toolbar in a given location (relative to the other toolbar icons on that toolbar). The following properties are significant in a Toolbar Item object definition:

- **Command.** Name of the Command object definition that is to provide the bitmap, method and target for the toolbar item. One or more hyphens can be specified instead of the name of a Command object to tell the system to insert a separator there between icons.

- **HTML Type.** Identifies the type of control to be displayed in the toolbar in the browser. Options include ComboBox, Button, Edit, Label, Hyperlink, MiniButton and Timer.
- **Name.** Name of the toolbar item. Used internally in Siebel Tools only. This needs to be unique within the scope of a toolbar.
- **Sequence.** Integer that orders the toolbar item in the parent toolbar from left to right. A higher sequence number relative to other toolbar items places this icon further to the right than the others.
- **Position.** Used for sideways toolbars. A value of .1, .2, and so on, is used.
- **Menu.** Defines a set of application-level main menus. Currently the only Menu object definition in use is called “Generic.” It is the parent of Menu Item.
- **Menu Item.** Defines an application-level main menu or menu item within the parent Menu object definition. Significant properties are the following:
 - **Name.** Uniquely identifies the menu or menu item.
 - **Command.** Name of the Command object definition that is to provide the method and target for the menu item.
 - **Caption.** The text displayed in the menu or menu item.
 - **Position.** Specifies the position of the menu or menu item in the menu tree. The top-level positions for menus (rather than items within them) are single integers such as 1, 2, and 3 (or 10, 20, and 30). Position values for menu items (2nd level and below in the menu tree) are specified using a dot notation, where the value to the right of the rightmost dot specifies the order of the item on its level, and the value or values to the left of that dot specify the parent menu or menu item. For example, 3.1 is the first item in the submenu of the third item of the top-level menu. Note that values on a level are not required to be consecutive; for example, the values 1110, 1115, 1120, and 1130 may be used to indicate four menu items on the same level; their sequence determines their order of display in the menu.

Applet Method Menu Item

Defines a menu item in the applet-level menu for the parent Applet object definition. Important properties are the following:

- **Menu Text.** The text displayed in the menu item.
- **Suppress Menu Item.** Default is FALSE. If TRUE, causes the class-level menu item of the specified name to be removed from the applet-level menu in the applet where this property is specified.
- **Command.** Name of the Command object definition that is to provide the bitmap, method and target for the applet menu item.
- **Position.** The sequence of the menu item in the single-level list of menu items.

See [“Defining Web Menus Using the Applet Method Menu Wizard” on page 591](#) for more information.

Class Method Menu Item

Class Method Menu Item is a child of Class. It adds (or suppresses) a menu item on applet-level menus for all SWE applets of the specified applet class and its subclasses.

Significant properties are the following:

- **Target.** Specifies which entity handles the invoke method specified in the Method property. Available options are the following:
 - **Browser.** The method handler is a JavaScript service on the browser, or the JavaScript applet class (secondarily the JavaScript business component class) on the browser, depending on whether a service is specified in the Business Service property.
 - **Server.** The method handler is an object manager service on the server or the applet and business component and their superclasses, depending on whether a service is specified in the Business Service property.
- **Menu Text.** The text displayed in the menu item.
- **Method.** The method invoked when the item is selected.
- **Business Service.** If specified, identifies the service on which to invoke the method. If unspecified, the method is invoked on the applet class on the browser or server (as specified in the Target property) with subsequent retargeting if unhandled.

- **Suppress Menu Item.** Default is FALSE. If TRUE, causes the applet-level menu items of the specified name to be removed from the applet-level menu in all applets derived from this class and its subclasses.
- **Position.** The sequence of the menu item in the single-level list of menu items.

Activating and Suppressing Menu Items and Toolbars

Menu items (both application-level and applet-level) and toolbar items can be activated or disabled at runtime, by means of the `CanInvokeMethod` mechanism. `CanInvokeMethod` (for the method specified in the `Command`, `Applet Method Menu Item`, or `Class Method Menu Item` object) will be called automatically for each item prior to displaying the menu or toolbar. If `CanInvokeMethod` returns FALSE, the toolbar item or menu item is not displayed. The `CanInvokeMethod` logic in most cases is retargeted from the browser application to the applet class hierarchy on the server, and from there to the business component class hierarchy. The targeting sequence is described below in `Invoke Method Targeting`.

Suppression and activation of individual applet-level menu items at design time is supported by means of the `Suppress Menu Item` property in the `Class Method Menu Item` and `Applet Method Menu Item` object types. This is applicable to applet-level menus only, not application-level menus or toolbars, in which the item must be added or removed explicitly in Tools. Design-time menu activation or suppression for applet-level menus provides the means to make a menu item available globally for applets of a given class and its subclasses, and then suppress it in particular applets where it is not desired. Certain applet-level menu items appear in virtually all applets (such as Copy, Edit, and Delete), others appear in virtually all list applets (such as Columns Displayed), and so on, but there are always exceptions in which a "standard" menu item for the applet's class needs to be suppressed for a specific applet.

To add applet-class-level menu items, you would add a Class Method Menu Item for a standard menu item for a given applet class. This menu item would not need to be re-included as Applet Method Menu Item object definitions in applets where you want the menu item to appear. You would only create Applet Method Menu Item object definitions in two circumstances: to add a menu item (not already provided by the applet's class) to the applet, or to suppress display of an applet-class-level item that the applet would normally inherit. In this latter case, you create an Applet Method Menu Item object definition with the same name as the applet-class-level menu item you want to suppress, and enter a value of FALSE for the Suppress Menu Item property.

Invoke Method Targeting

The Method, Business Service, and Target properties appear in the Command object type for use in toolbars, application-level menus, and applet menus. The target property specifies the object or service that will process the method invoked by the command. Under some circumstances, if a method cannot be handled by the specified target it is automatically directed to an underlying object or service for handling. This could be a mirror instance of the object that exists on the server rather than the browser, or it could be an inherited class. In these cases we say that the method invocation has been retargeted.

Two settings are available for the Target property, with the following behavior:

- **Browser target.** The method handler for this target is the JavaScript application, a JavaScript applet, or a JavaScript service, on the browser side. In all cases, a method name must be specified in the Method property. A service is targeted if a service name is specified in the Service property. If a service is not specified, method handling differs based on whether the calling entity is application-level or applet-level, as follows:
 - **Application-level.** Targets to the specified method in the JavaScript application. Does not retarget.
 - **Applet-level.** Targets to the specified method in the JavaScript applet. If not handled, retargets to the specified method in the corresponding JavaScript business component. No inheritance or additional retargeting.

- **Server target.** This target is for invoking a method in a C++ class on the server, either on a service or on the infrastructure. If a Service property value is not specified, the invoke method is targeted to the infrastructure. It will target the infrastructure differently depending on whether the menu or toolbar icon invoking the method is applet-level (menu only) or application-level (menu or toolbar).
- **Application-level.** The method handler is initially the SWE UDF loader on the server side, and secondarily the model.
- **Applet-level.** The method handler is initially the applet class to which the applet belongs, and is retargeted successively up through the applet class hierarchy to CSSSWEFrame. If still unhandled, handling is retargeted to the business component class of the applet's business component, and successively upwards through the business component class hierarchy to CSSBusComp.

If a service is specified in the Service property, the method handler is the specified service. This targeting is also dependent on whether the calling menu item or toolbar icon is applet-level or application-level, as follows:

- **Application-level.** The method handler is the specified OM service. It does not retarget.
- **Applet-level.** The method handler performs a SetBC call to set to the business component of the applet, and then calls the specified OM service. It does not retarget.

The results of the possible settings of the Target and Business Service properties at the applet and application levels are summarized in [Table 38 on page 523](#).

Table 38. Target and Business Service Properties Matrix

Menu/Toolbar Level	Target	Service	Result
Application level	Server	Specified	The method handler is the specified business service on the server. It does not retarget.
		Unspecified	The method handler is the base functionality associated with an application object.
	Browser	Specified	Targets to the method in the specified browser-side service. It does not retarget.
		Unspecified	Targets to the specified method in the JavaScript application. It does not retarget.
Applet level	Server	Specified	The method handler calls the specified service on the server. It does not retarget.
		Unspecified	The method handler is initially the applet class to which the applet belongs, and is retargeted successively up through the applet class hierarchy to CSSSWEFrame. If still unhandled, handling is retargeted to the business component class of the applet's business component, and successively upwards through the business component class hierarchy to CSSBusComp.
	Browser	Specified	Targets to the method in the specified browser-side service. It does not retarget.
		Unspecified	Targets to the specified method in the JavaScript applet. If not handled, retargets to the specified method in the corresponding JavaScript business component. There is no inheritance or additional retargeting.

Icon Maps

The *Icon Map* object type allows you to render control or list column field values as icons. Each Icon Map is a collection of child objects called Icons. Icon objects are associated with a Bitmap object, which defines the image for the Icon and corresponds to a particular field value. Controls and list columns have an attribute called Icon Map that allows you to define the icon map object that you want to use for rendering the field values.

The following procedure uses an example to show you how to configure icons for use as field values. The example uses the Status list column on the Activity List Applet. Suppose that the Status field can have values Not Started, In Progress, and Done. You want to configure the Status field to display an icon for each of these values.

To render fields using Icons

- 1 Create a Bitmap Category.

For example, create a Bitmap Category called Activity Status Icons.

- 2 Create Bitmaps (child object of Bitmap Category) for each image that you want to display and specify the file name of the image.

For example, create the following records:

Name	File Name
Not Started	notstarted.gif
In Progress	inprogress.gif
Done	done.gif

- 3 Create a new Icon Map object.

For example, create an Icon Map named Activity Status.

- 4 Create one Icon object (child of Icon Map) for each field value and set the following properties:

- **Name.** Set to the name of the field value.

- **Bitmap Category.** Set to the Bitmap Category you want to show for the field value.
- **Bitmap.** Set to the bitmap you want to show for the field value.

For example, create the following records:

Name	Bitmap Category	Bitmap
Not Started	Activity Status Icon	Not Started
In Progress	Activity Status Icon	In Progress
Done	Activity Status Icon	Done

- 5 Set the HTML Icon Map attribute of the list column or control to the Icon Map defined in [Step 3](#).

For example, set the Status list column of the Activity List Applet to the icon map Activity Status.

After compiling the changes, the Siebel Web Engine will render the image corresponding to the bitmap when the field value matches one of the icons defined. If the field value does not match any of the icons, the Siebel Web Engine renders the field value itself.

Specifying a Default Icon in an Icon Map

You can create an icon named Default in a Icon Map object. If the field value does not match any of the icons, then the Default icon is used for the field. This feature is useful to create an icon to be used with fields that could contain different values, such as URLs. In this case you would set the HTML Type property of the field to be URL and its IconMap property to an IconMap object that contains only one icon named Default.

HTML Hierarchy Bitmap

An *HTML Hierarchy bitmap* is a top-level object that defines the images used by hierarchical objects such as a Tree applet when it is rendered in the user interface. For example, the folders, the plus symbol, and the minus symbol in [Figure 157](#) are icons defined in the Hierarchy Bitmap object. An example of a Tree Applet in the repository is an Account Tree Applet. An example of an hierarchical list applet is Quote Item List Applet.

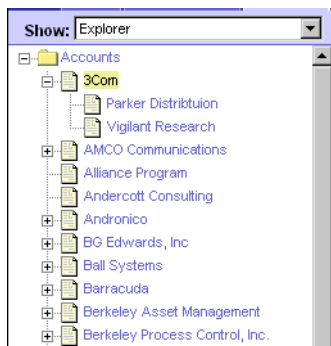


Figure 157. Tree Portion of the Account Tree Applet

Hierarchy Bitmap objects have the following important properties:

- **Name.** The name for the HTML Hierarchy Bitmap object.
- **Collapse Bitmap, Collapse Elbow Bitmap, Collapse Tee Bitmap.** Icons to be used to collapse a node.
- **Expand Bitmap, Expand Elbow Bitmap, Expand Tee Bitmap.** Icons to be used to expand a node.
- **Elbow Bitmap, Tee Bitmap.** Icons to be used for creating an elbow (L) or a Tee (T).
- **Bar Bitmap.** Icon for creating a vertical line.
- **Space Bitmap.** Icon for indents.
- **Open Bitmap.** Icon to be used for a node that is an expanded state.

- **Close Bitmap.** Icon to be used for a node that is in a collapsed state.
- **Leaf Bitmap.** Icon for a leaf node.
- **Arrow Down Bitmap, Arrow Up Bitmap.** Icons for scrolling a tree up or down.

The bitmap objects for these attributes used by the standard tree applets are defined in the Bitmap Category HTML Hierarchy Icons.

The Tree and List objects in tools have an attribute called HTML Hierarchy Bitmap. This attribute can be set to the name of any HTML Hierarchy Bitmap object. This allows the various instances of the Tree object and List object to share these bitmaps.

The Tree Node object has the attributes HTML Open Bitmap and HTML Close Bitmap. These attributes are optional. If you do not specify, the Open Bitmap and Close Bitmap attributes of the HTML Hierarchy Bitmap object will be used. If the attributes are specified, then for that node the specified attributes will be used. This is useful if you want different nodes to have different icons.

See [“Hierarchical List Applets” on page 898](#) for more information.

Logical User Interface Objects Layer

HTML Hierarchy Bitmap

This chapter describes how to define objects in the user interface layer.

User Interface Object Definition Sequence

Follow this sequence when you create objects in the User Interface Object Layer.

Before you begin working on user interface objects:

- Check out the relevant project.
- Define the required objects in the Business Objects Layer.

To create the objects in the User Interface Objects Layer

- 1** Define applets, as well as the required controls and list columns.
See [“About Defining Applets” on page 530](#) for more information.
- 2** Map list columns and controls to Web templates.
See [“Editing the Web Layout of Applets” on page 544](#) for more information.
- 3** Define a view.
See [“Creating Views Using the View Wizard” on page 563](#) for more information.
- 4** Map applets to view Web template.
See [“Editing the Web Layout of Views” on page 566](#) for more information.
- 5** Define a screen and Add the view to the screen.
See [“About Screens” on page 572](#) for more information.

- 6** Associate the screens to the page tabs and menu items.
See [“About Screens” on page 572](#) for more information.
- 7** Create an application (or use an existing one).
See [“About Applications” on page 576](#) for more information.
- 8** Validate your business object definitions.

About Defining Applets

Applets provide viewing, entry, modification, and navigation capabilities for data in one business component.

Applets appear as part of a view and are typically implemented as a list table or data entry form. Applets can be populated with standard controls like buttons, text boxes, check boxes, and custom controls. Applets can also include ActiveX controls, which are externally created program units that can interact with the applet through property settings, methods, and events.

Some applets perform specialized roles as dialog boxes, charts, and trees. For more information about applet styles and types, see [Chapter 9, “Logical User Interface Objects Layer.”](#)

Figure 158 shows the object definition for a list applet called Opportunity.

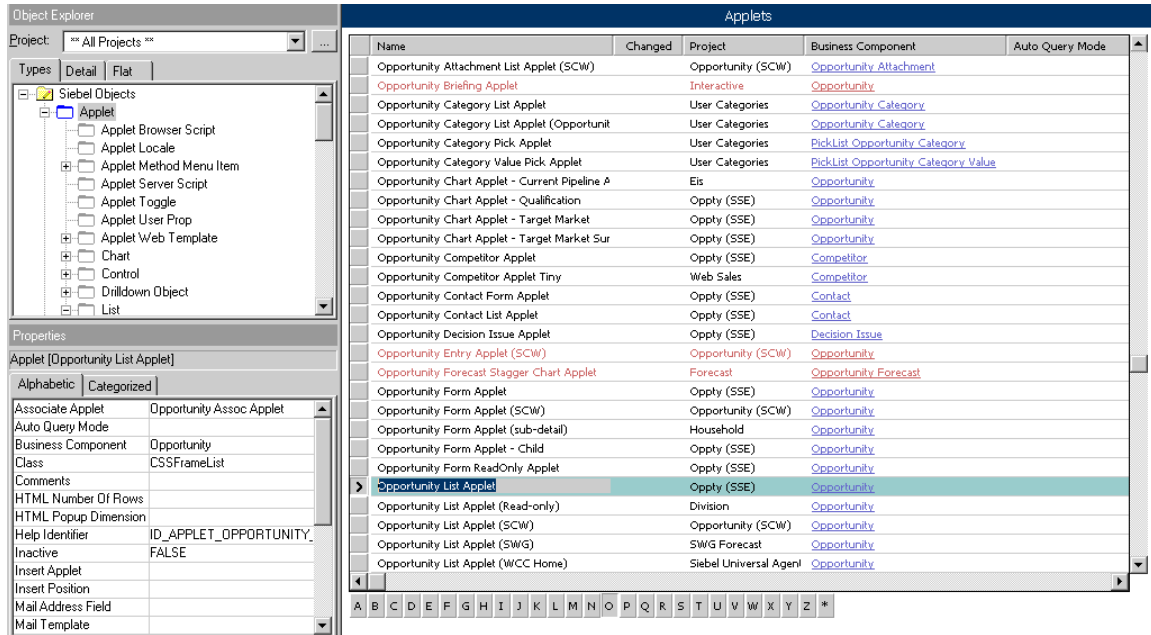


Figure 158. Opportunity List Applet

Typically it is easier to modify an existing applet than create a new one. However, in some cases it may be necessary to add a new applet using the steps shown in [Figure 159](#).

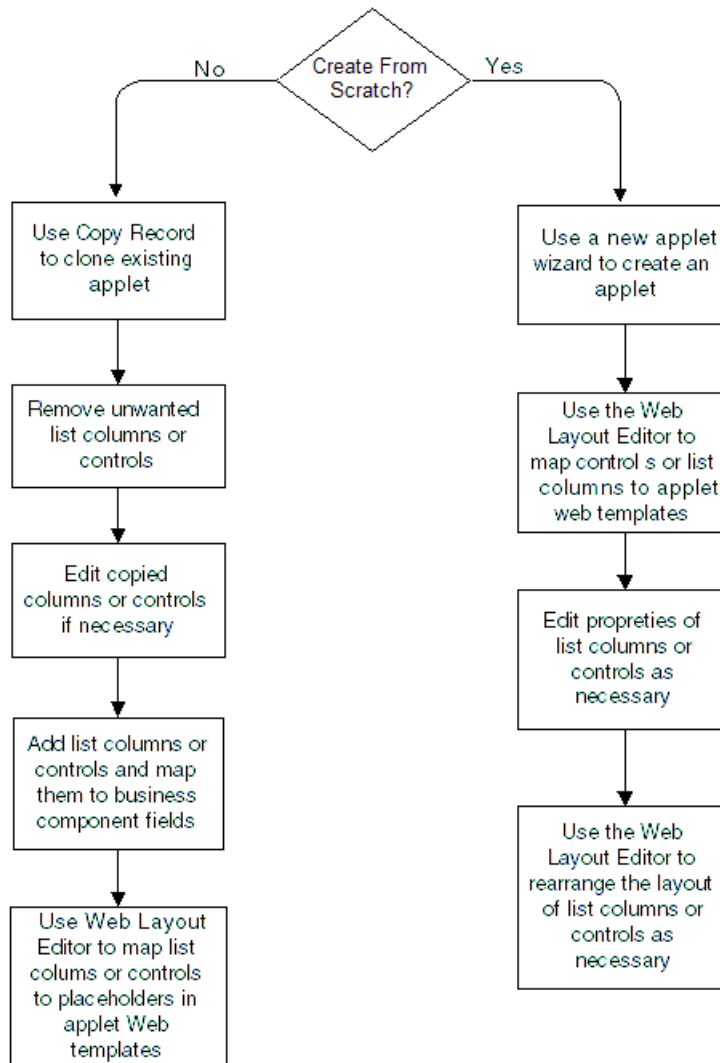


Figure 159. Defining List Applets

About Applet Properties

Following are descriptions for key applet properties. For a complete list and more detailed descriptions, see *Object Types Reference*.

- **Business Component.** (Required.) The name of the business component whose data is to be displayed in the applet.
- **Class.** (Required.) The C++ class used to manage the applet. Form applets use the CSSFrame class, and list applets use the CSSFrameList class. There are also some specialized classes for applets that use special business components and have special features.

CAUTION: Controls on applets belonging to a specialized class should not be deleted because this may break methods used on the applets.

NOTE: Do not change the Class property of previously configured applets.

- **Name.** (Required.) Name of the applet.
- **No Delete, No Insert, No Update.** If TRUE, you are not allowed to perform the data manipulation operation.

NOTE: If these are true for the business component on which the applet is based, the applet will automatically inherit the restrictions from the business component.

- **Search Specification.** A conditional expression used to restrict the records displayed.
- **Title.** The text used for the title of the applet.

Applet Controls

Following are descriptions for key applet Control properties. For a complete list and more detailed descriptions, see *Object Types Reference*.

List Applet Control Properties

The following are the key control properties for all list applets:

- **List.** (List Control.) Control used for displaying columns of data from the underlying business component.

Accept the default values for these list applet controls.

Form Applet Control Properties

- **Caption.** (MiniButton, CheckBox, Group, Label Controls.) Display value that identifies the purpose of the control to the end user. The caption is typically rendered as a label within or next to the control.
- **Field.** (TextBox, CheckBox.) Business component field for which the control is displaying data.
- **Method Invoked.** (PushButton.) Method invoked when the button control is pushed.
- **Name.** (Required.) The name of the control.
- **Read Only.** (TextBox, CheckBox.) If TRUE, the control is read-only and the value in it cannot be modified.
- **HTML Sequence.** The position of this control relative to other controls in the applet, indicating tab sequence. Valid values are numbers greater than zero.
- **Text Alignment.** (Label Controls.) Left is the default. It indicates how to align the text in the control.
- **HTML Type.** The control type.

NOTE: Controls on applets belonging to a specialized class should not be deleted because it may break methods used on the applets.

Defining List Applets

You create list applets using the List Applet Wizard. The List Applet Wizard helps you identify all the correct properties and automatically creates child objects, such as Web Template Items, based on the information you enter. You can also create applets manually by defining all the necessary properties and child objects.

The List Applet Wizard does the following:

- Creates the list applet
- Creates the applet Web template
- Creates the list, list columns, and controls
- Creates applet Web template items

To create a list applet using the List Applet Wizard

- 1** Select File > New Object from the Siebel Tools main menu.

The New Object Wizard dialog box appears.

- 2** Click the Applets tab, and then double-click the List Applet icon.

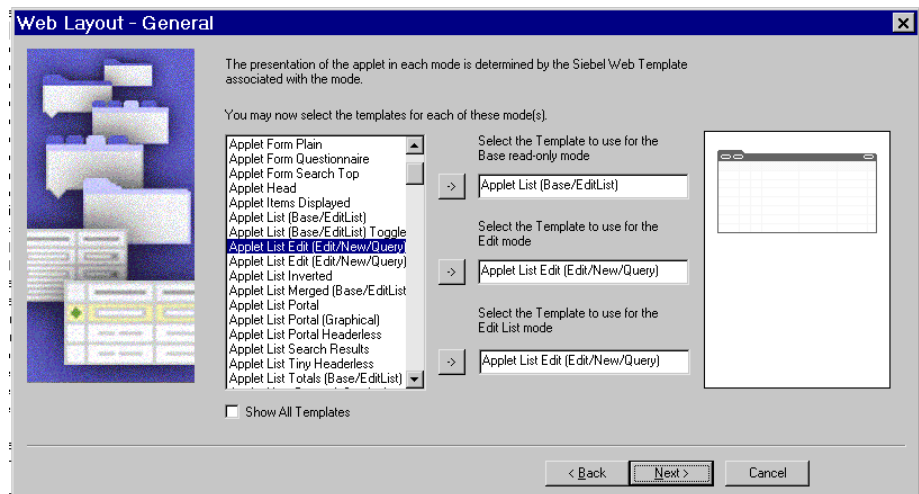
The General page of the List Applet Wizard appears.

- In the General Page, enter the following information for the applet, and then click Next.

Field	Example Value	Comment
Project	Account	Only locked projects appear in the picklist.
Business Component	Account	The business component that the applet is based on.
Applet Name	New Account List Applet	A unique name for the Applet.
Display Title	Accounts	The name to appear in the user interface.

The wizard will use information to create an applet object and define the required applet properties.

- In the Web Layout-General page, enter the Web templates to use for the applet, and then click Next.



NOTE: A thumbnail image for most templates appears when you select the template name. For a complete description of all templates, see *Siebel Developer's Reference*.

- 5** In the Web Layout - Fields page, select the fields that you want to appear on the applet, and then click Next.

The fields that appear in the Available pane are those fields defined for the business component that you selected in [Step 3 on page 537](#).

- 6** In the second Web Layout-Fields page, choose the controls in the Available Controls box that you want to appear on the applet, and then click Next.

All the entries in the Selected Controls box are added by default. If you wish to exclude some of the controls and move them to the Available Controls box, select the controls and click the activated arrow.

NOTE: The available controls come from the Model HTML Controls Applet. This applet specifies the available controls and also to which template each control is mapped. Users can modify this applet if necessary by adding or removing controls from the applet.

- 7** Review the information displayed in the Finish page, and then click Finish.

The List Applet Wizard creates the applet and supporting object definitions based on the selections you made.

NOTE: You can return to previous pages by clicking the Back button.

Defining List Columns in a List Applet

Figure 160 shows list columns mapped to fields in the Opportunity business component.

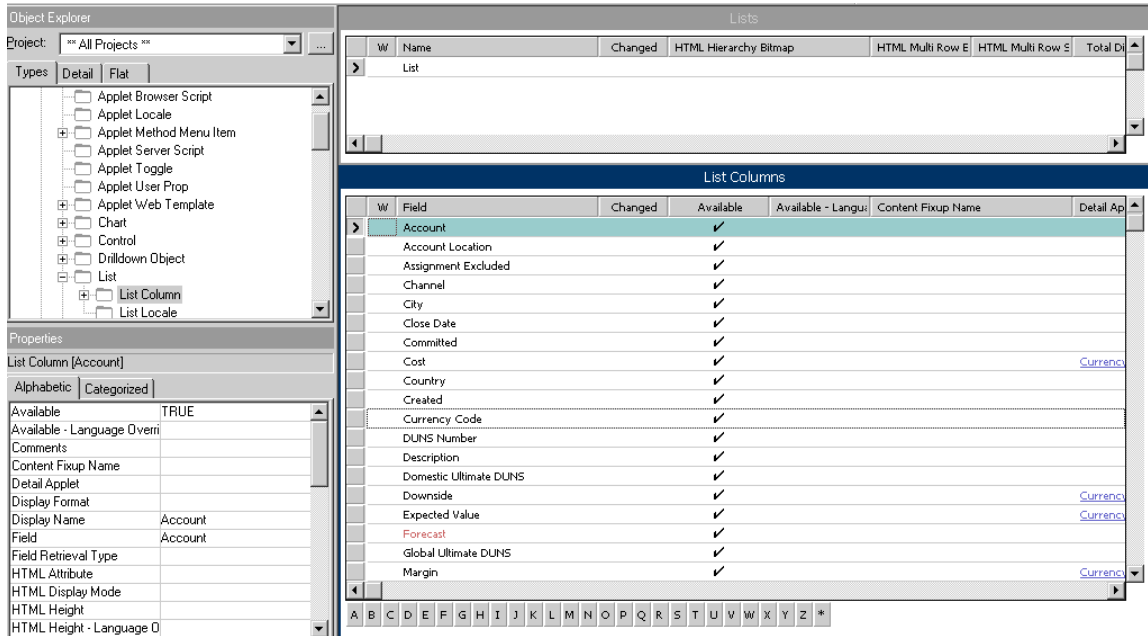


Figure 160. Field List Columns in Opportunity Business Component

End User Settings and Preferences

End users can choose which list columns to display, and the order in which they are displayed in the Siebel client. These settings override Siebel Tools configuration settings.

To test a configuration without user-specified settings

- In the Siebel Web client, go to the list applet to which your completed configuration changes apply, click the menu button, select Columns Displayed, and then click the Reset Defaults button. Inspect the list applet to be sure your configuration changes are what you expect.

Lists and List Columns

A list applet contains only list columns, or list columns and buttons.

List is the parent of the List Column object type. A list applet can contain only one List object definition.

HTML Type Property of List Columns and Controls

The HTML Type property of a list column or control determines the type of control. Examples of HTML Type controls are, Check Box, Combo Box, Text, MiniButtons, Password, PositionOnRow, and URL.

For more complete descriptions of each control type, see [“About HTML Control Types” on page 479](#).

Enhancing the Behavior of List Columns and Controls

For Text controls and list columns:

- If a calendar or calculator needs to appear for control, set the Runtime property of the control to TRUE.
- If a pop-up editor needs to appear, set the Popup Edit property of the control to TRUE.
- If you want to make a list column unavailable in the user interface, set the Available property to FALSE.
- If you want a list column to be displayed in the list by default, set the Show in List property to TRUE. However, if you want the list column to be hidden by default, but you want end users to be able to select a list column to be displayed using the Columns Displayed dialog box, set the Show in List property to FALSE.

Defining Form Applets

You create form applets using the Form Applet Wizard. The Form Applet Wizard helps you define the correct properties and automatically creates child objects, such as Web Template Items, based on the information you enter. You can also create applets manually by creating all the necessary objects and defining object properties.

The Form Applet Wizard does the following:

- Creates the form applet
- Maps the applet to an applet Web template
- Creates the controls
- Maps controls to Web templates by creating Applet Web Template Items

To create a Form Applet using the Form Applet Wizard

- 1** Choose File > New Object from the Siebel Tools main menu.

The New Object Wizard dialog box appears.

- 2** Click the Applets tab, and then double-click the Form Applet icon.

The General page of the Form Applet Wizard appears.

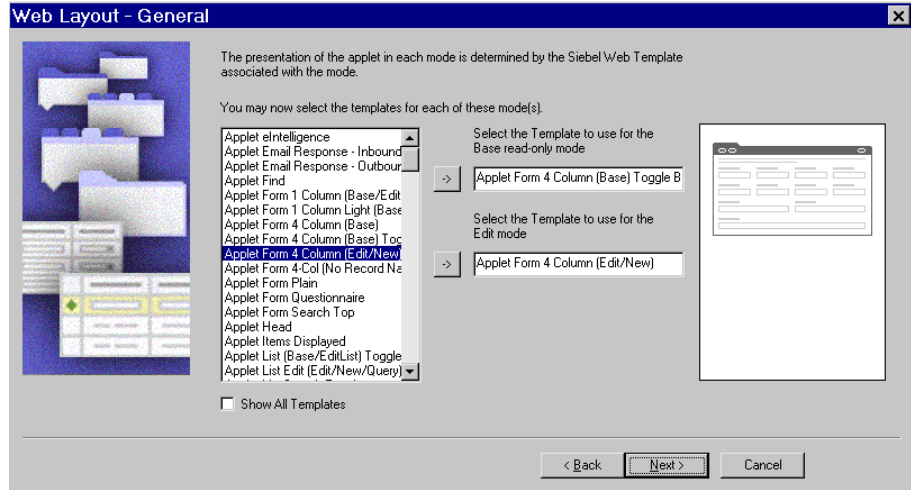
- 3 In the General Page, enter the following information for the applet.

Field/Checkbox	Example Value	Comment
Project	Account	Only locked projects appear in the picklist.
Business Component	Account	The business component that the applet is based on.
Applet Name	New Account Form Applet	A unique name for the Applet.
Display Title	Accounts	The name to appear in the user interface.
Use Grid Layout	Check mark	Select this option if you want to have the ability to control the layout of the form applet using the Web Layout Editor. For more information see “About Grid Layout” on page 547 .

The wizard will use information to create an applet object and define the required applet properties.

If you selected Use Grid Layout, the appropriate grid layout Web template is associated with the template and you can skip the next step and go to directly to [Step 5](#).

- 4 If you did not select the Use Grid Layout option In the Web Layout-General page, enter the Web templates to use for the applet, and then click Next.



For a complete description of templates, see *Siebel Developer's Reference*.

- 5 In the Web Layout - Fields page, select the fields that you want to appear on the applet and then click Next.

The fields that appear in the Available pane are those fields defined for the business component that you selected in [Step 3 on page 537](#).

- 6 In the second Web Layout-Fields page, choose the controls in the Available Controls box that you want to appear on the applet and then click Next.

All the entries in the Selected Controls box are added by default. If you wish to exclude some of the controls and move them to the Available Controls box, select the controls and click the activated arrow.

NOTE: The available controls come from the “Model HTML Controls” applet. This applet specifies the available controls and also to which template each control is mapped. Users can modify this applet if necessary by adding or removing controls from the applet.

- 7 Review the information displayed in the Finish page, and then click Finish.

The Form Applet Wizard creates the applet and supporting object definitions based on the selections you made.

NOTE: You can return to previous pages by clicking the Back button.

Editing the Web Layout of Applets

You edit the web layout of applets using the Web Layout Editor. The Web Layout Editor allows you to perform the following tasks for all applets:

- Add, delete, and position controls
- Modify Captions for controls and Display Names for list columns
- Configure a control to be hidden by default and to appear when the user clicks the More button on the applet
- Preview the applet layout as it will appear in the user's Web Browser at runtime
- Export the preview to an HTML File

For applets based on Grid Layout templates, the Web Layout Editor also allows you to perform additional formatting tasks. See [“About Grid Layout” on page 547](#) for more information.

To modify the layout of applets

- 1 Select a Target Browser from the drop-down list in the Configuration Context toolbar.

If you do not select a browser, an error message appears when you open the Web Layout Editor.

- 2 In the Object Explorer, select the Applet object type.
- 3 In the Object List Editor, select an applet.

- 4 Right-click, and then choose Web Layout Editor.

The Web Layout Editor appears.

NOTE: If an applet Web template is not yet associated with the applet you selected, a dialog box appears that allows you to open Applet Wizard to associate a web template with the applet. See [“Defining List Applets” on page 536](#) or [“Defining Form Applets” on page 540](#) for more information.

- 5 In the Mode field of the Configuration Context toolbar, choose the applet Web template that you want to edit.

Make sure that you select an active Web template. Both active and inactive Web templates are displayed.

Changes to an applet layout in one mode are not automatically propagated to the applet layout in another mode.

- 6 Perform the following tasks as necessary:

Task	Action
Add existing controls and list columns	Drag a control or list column from the Control/Column window and drop it onto the applet layout. In applets based on Grid Layout applet Web templates, labels and controls are treated as separate items to provide more flexibility when designing the form layout. However, this requires you to map both the control and its corresponding label onto the applet layout. Note that a label has the same name as its corresponding control, except that it is appended with the word <i>label</i> . See “About Grid Layout” on page 547 for more information.
Add new controls and columns	Drag and drop new controls from the Web Controls toolbar. The control is automatically created as a child object of the applet. After you save your changes, you can define the properties for the control using the Properties Window. To display the properties window, select the control, right-click, and then choose View Properties Window.

Task	Action
Position controls	Drag and drop controls to different locations on the web layout canvas. For applets based on column-based Web templates, the available locations are determined by placeholders in the Web template. For applets based on Grid Layout Web templates, you can position controls and their corresponding labels anywhere in the canvas. See “Editing the Web Layout of Applets” on page 544 . Note that placeholders in the banner region across the top of applets are indented for use with button controls only, not fields.
Delete controls	Select the control you want to delete, right-click, and then select Delete. You can select multiple controls by holding down the Ctrl key and selecting the controls you want to delete.
Edit display names and captions	<p>For column-based applet Web templates, double-click a control, highlight the display name or caption, and then type new text. After you save your work, the Display Name property for list columns and the Caption property for controls are automatically changed.</p> <p>For grid-based templates, you cannot change the Caption value of a label. Labels get their property values from their corresponding controls. See “Editing the Web Layout of Applets” on page 544 for more information about labels and controls in a grid-based applet Web template.</p>
Display a control only after the Show More button has been clicked	Select the control, right-click, and then choose More. An arrow is displayed next to the control and label. The arrow indicates that the control will appear only after the user selects the Show More button. You can select multiple controls by holding down the Ctrl key and selecting the controls.
Preview the Web layout	Right-click, and then choose Preview. The applet is displayed as it would be rendered in the user interface at runtime.

Task	Action
Export the preview to an HTML file	Select File > Export, and then in the dialog box that appears, choose a file name and define <i>siebel_tools_installation\tools\public\enu</i> as the directory. You need to specify this directory so that image files, such as buttons, are rendered in the HTML file.
Check mappings—validate that controls and list columns are mapped to placeholders in the Web template	In the Web Layout Editor, right-click, and then select Check Mappings. If a control or list column is not mapped to a placeholder, you are prompted to delete it from the Web template. This may occur when a control or list column object has been deleted from the repository, but still appears on the Web template. Or it may occur when a new Web template is associated with an applet, and the existing placeholder IDs for controls or list columns do not exist in the new template.

- 7 Save your changes to the Web Layout by choosing File > Save.

After you save your changes to the Web layout, objects for any new controls are created. You can then define object properties, such as Field and Name, using the Properties window.

About Grid Layout

Grid Layout is a set of visual design features in the Web Layout Editor and supporting applet Web templates that allow you to modify form layout without having to directly modify applet Web templates. The work space is a grid-based canvas where controls snap to a grid of cells that measure 8 x 8 pixels each. You can configure the layout of form applets using a palette of layout tools, such as resizing, aligning, and centering. See [“Supported Applet Classes and Applet Web Templates” on page 554](#) for more information about performing these tasks.

To support the Grid Layout features in the Web Layout Editor, applets must be based on one of the grid layout type applet Web templates. For more information about Grid Layout applet Web templates, see [“About Grid Layout Templates” on page 900](#).

When creating new form applets you can base them on the grid-based templates and use the Grid Layout features in the Web Layout Editor. For more information, see [“Defining Form Applets” on page 540](#).

Applets shipped with the Siebel repository are not based on grid-based applet Web templates. To be able to use the Grid Layout features with these applets, you must convert them to Grid Layout. For more information, see [“Converting Applets to a Grid-Based Layout” on page 548](#).

NOTE: You can determine if an applet uses a grid-based applet Web template by looking at the Web Template property of the Applet Web Template object type (child object of Applet). If it is a grid-based applet Web template, *Grid Layout* will appear in the name.

Converting Applets to a Grid-Based Layout

Applets shipped with the Siebel repository are not based on Grid Layout Web Templates. To be able to use the Grid Layout features, you must convert an existing applet to a grid-based layout using one of the methods covered in this section.

This section describes the following topics:

- [“Converting One or More Applets.”](#)
- [“Converting Applets By Changing the Web Template” on page 550.](#)
- [“About Grid Layout Conversion Error Messages” on page 552.](#)
- [“Supported Applet Classes and Applet Web Templates” on page 554.](#)

Converting One or More Applets

The Applet Web Template Conversion Wizard allows you to convert one or more existing applets to a grid-based layout.

To convert one or more applets to a grid-based layout

- 1** In the Configuration Context Toolbar, make sure the application context that you want, such as All Applications, is selected.

The wizard only converts controls that are valid in the current application context selected in the Configuration Context Toolbar. For example, if Siebel ERM is selected, only controls valid in the context of Siebel ERM are converted. If controls are not valid in the selected application context, a dialog box appears giving you the option of canceling the conversion or continuing. If you choose to continue, the controls not converted are written to a log file (`awtconversion.txt`) that is located in `tools_install\temp`. See [“About Grid Layout Conversion Error Messages” on page 552](#).

- 2** In the Object Explorer, select the Applet object type.
- 3** In the Object List Editor, select the applet or applets that you want to convert.

The applet must be based on one of the supported applet classes and one of the applet Web templates listed in [“Supported Applet Classes and Applet Web Templates” on page 554](#).

- 4** Do one of the following:
 - Right-click and then select Convert to Grid Layout.
 - Select Tools > Convert to Grid Layout.

The Applet Web Template Conversion Wizard appears.

- 5** Move the applets you want to convert from the Available Applets window to the Selected Window.
- 6** Select additional options.

It is recommended that you select the Backup existing Applet Web Templates option.

If you select the option to Display the Web Layout Editor after conversion completes, the applet web template displayed in the editor is the last applet that was selected in [Step 5](#).

7 Click Next.

The wizard converts the applets from standard Web templates to grid-based layout Web templates.

If no errors occur, you can edit the layout of these applets using the Web Applet Editor. For more information, see [“Editing the Web Layout of Applets” on page 544](#).

If errors occur, they are displayed in a dialog box at the end of the conversion wizard. The same content is stored in a log file (`awtconversion.txt`) that is located in `tools_install\temp`. For more information about errors, see [“About Grid Layout Conversion Error Messages” on page 552](#).

Converting Applets By Changing the Web Template

You can also convert applets to a grid-based layout by changing the Web template associated with the applet. In Siebel Tools, change the Web template file associated with each applet mode to a template that supports a grid-based layout.

There are two applet Web templates that support grid layout. See [Table 39](#).

Table 39. Grid Layout Templates

Web Template	File Name	Comments
Applet Form Grid Layout	CCAppletFormGridLayout.swt	Use with all modes of form applets.
Applet Popup Form Grid Layout	CCAppletPopupFormGridLayout.swt	Use with all modes of popup form applets.

See [“About Grid Layout Templates” on page 900](#) for more information.

To convert an applet to grid-based layout by changing the Web template

- 1 In the Object Explorer, select the Applet object type.

- 2** In the Object List Editor, select the applet that you want to convert to a grid-based layout and then right-click and choose Edit Web Layout.

The applet must be based on one of the supported applet classes and one of the applet Web templates listed in [“Supported Applet Classes and Applet Web Templates”](#) on page 554.

- 3** In the Web Controls toolbar, click the Change Template button.

If the Web Controls toolbar is not displayed, choose View > Toolbars > Web Controls to display it.

The Choose Template dialog box appears.

- 4** In the Choose Template dialog box, select one of the following templates:

For form applets, select Applet Form Grid Layout.

For pop-up form applets, select Applet Popup Form Grid Layout.

- 5** Repeat [Step 4](#) for each applet mode.

After associating a grid layout template with each mode of an applet, you can edit the applet layout using the Web Layout Editor.

For general instruction on editing the layout of applets, see [“Editing the Web Layout of Applets”](#) on page 544.

For information on editing applets using grid-based layout features, see [“Supported Applet Classes and Applet Web Templates”](#) on page 554.

About Grid Layout Conversion Error Messages

The error messages listed in [Table 40](#) may appear during the conversion. These errors appear in a dialog box at the end of the conversion process and are written to a log file (`awtconversion.txt`) that is located in `tools_install\temp`.

Table 40. Conversion to Grid Layout Error Messages and Solutions

Error Message	Cause	Solution
Controls or labels cannot be mapped.	May be due to the web template item not being explicitly mapped to a control on the original applet web template. Each web template item must have the Control property populated in order to appear on the new grid-based applet web template.	Use Web Layout Editor to map controls to the applet layout. See “Editing the Web Layout of Applets” on page 544 .

Table 40. Conversion to Grid Layout Error Messages and Solutions

Error Message	Cause	Solution
<p>Applet cannot be converted.</p>	<p>Applet class or associated Web templates are not supported for Grid Layout. See “About Grid Layout” on page 547.</p>	<p>The applet and web template combination you are trying to convert is not currently supported. You can submit an enhancement request to Siebel Technical support requesting that the class or template be supported for future releases.</p>
<p>The Applet Web Template...is configured for more than one application context.</p>	<p>The wizard only converts controls that are valid in the application context selected in the Configuration Context Toolbar at the time of the conversion. For example, if Siebel ERM is selected, only controls valid in the context of Siebel ERM are converted. If controls are not valid in the selected application context, a dialog box appears giving you the option of canceling the conversion or continuing. If you choose to continue, the controls that are not converted are listed in a dialog box and are written to the error log file.</p> <p>Note that the Expression property of the Web Template Item object type determines the application context for a control.</p>	<p>Select the appropriate application in the Application field of the Configuration Context toolbar and run the Conversion Wizard again.</p> <p>If the configuration Context toolbar is not displayed, choose View > Toolbars > Configuration Context.</p> <p>For more information, see “Converting One or More Applets” on page 548.</p>

Supported Applet Classes and Applet Web Templates

Not all form applets can be converted to a grid layout. For version 7.5.3, an applet's class and its associated Web templates must be supported for conversion to a grid-based layout. This section lists the supported applet classes and applet Web templates at the time of publication. For the most current list, review the Applet Web Template Conversion Wizard's configuration file. The file name is `awtcvtcfg.txt` and the file location is `Tools_Install\BIN`.

NOTE: Editing the list of applet classes and applet Web template files defined in the Applet Web Template Conversion Wizard configuration file (`awtcvtcfg.txt`) is not supported.

Applet Classes

The applet classes supported for conversion to a grid-based layout are:

- `CSSFrame`
- `CSSFrameBase`
- `CSSFrameSalutation`
- `CSSSWFrameWeb`
- `CSSFrameQuote`
- `CSSSWFrame`
- `CSSSWFrameBase`
- `CSSFrameList`
- `CSSFrameListBase`
- `CSSFrameListWeb`
- `CSSFrameListFile`

NOTE: The applet classes with "List" in the title, such as `CSSFrameList`, are list applets with edit modes that are forms. The Edit mode of these applets can be converted to a grid-based layout.

Applet Web Templates

The applet Web templates supported for conversion to a grid-based layout are:

- Applet List Edit (Edit/New/Query)
- Applet List Edit (Edit/New/Query) Toggle Bar
- Popup Query
- Applet Form 4 Column (Edit/New)
- Applet Form 4 Column (Base)
- Applet Form 4 Column (Base) Toggle Bar
- Applet Form 4 Column (Edit/New) - Expanded
- Applet Form 4 Column (Base) - Expanded
- DotCom Applet Form 4-Column
- DotCom Applet Form 2-Column
- DotCom Applet Form 1-Column
- Applet Form 1 Column (Base/Edit/New)
- Applet Form 4-Col (No Record Nav)
- DotCom Applet Form Basic
- DotCom Applet Form Base 1 Column
- Applet Form Plain
- DotCom Applet Form 4-Column (No Record Nav)

Editing Applets Based on Grid Layout Templates

Grid layout features allow you to modify the layout of applets more easily than in previous versions. You can specify the layout of controls in form applets without having to modify Web templates. Only applets that are based on grid layout applet Web templates support these features. See [“About Grid Layout” on page 547](#) for more information.

The tasks in this section cover the layout and formatting for applets based on grid-based layout, including:

[“Positioning Controls” on page 557.](#)

[“Aligning Controls” on page 557.](#)

[“Resizing Controls” on page 558.](#)

[“Spacing Controls” on page 559.](#)

[“Centering Controls” on page 560.](#)

[“Setting Tab Order” on page 561.](#)

For general instructions about using the Web Layout Editor, see [“Editing the Web Layout of Applets” on page 544.](#)

When working with grid layout applets in the Web Layout Editor, consider the following:

- Controls snap to a grid in which each grid cell measures 8 x 8 pixels.
- Grid-based layouts do not automatically resize based on the user’s monitor settings. For example, if a grid-based form designed to run on a monitor set to a resolution of 1024 x 768 (full width) is displayed on a monitor that is set to 800 x 600, the user would have to scroll to the right to see the right edge of the layout. If the same form is displayed on a monitor set to a resolution of 1280 x 1024, it would not occupy the entire width of the screen.
- Labels appear as separate items from their corresponding controls or list columns. This gives you the ability to configure the layout of labels independently from their corresponding controls. However, labels do not exist as separate controls in the Siebel repository. They share the Caption or Display Name property of their corresponding control or list column, but other properties do not apply. Labels are used as constructs in the Applet Web Layout Editor only.

- By default, when you save an applet layout, the Web Layout Editor checks for overlapping controls. If overlapping controls exist, a dialog box appears giving you the option to save the layout or cancel. If you save the layout, with the Web Layout Editor open, the overlapping controls are highlighted. If you select cancel, the layout is not saved. You can control whether the Web Layout Editor checks for overlapping controls by choosing Tools > Options, and then setting the *Prompt when saving overlapping controls* option.
- When working in the Web Layout Editor, the last item selected on the canvas will be the one to which all other items will be aligned, centered, spaced, and so on.

Positioning Controls

Grid layout allows you to position controls in the applet layout.

To position controls

- 1 In the Web Layout Editor, select one or more controls.
- 2 Do one of the following:
 - Drag and drop the control or controls to the desired position.
 - Use the arrow keys to move the control or controls to the desired position.
- 3 Choose File > Save.

Aligning Controls







Grid layout allows you to align controls relative to each other.

To align controls

- 1 In the Web Layout Editor, hold down the Ctrl key, and then click the controls you want to align.

The last item selected is the one to which all other items will be aligned.

- 2 Perform the following tasks as necessary:

Task	Button
Align the left side of controls.	
Align the center of controls vertically.	
Align the right side of controls.	
Align the top edge of controls.	
Align the middle of controls horizontally	
Align the bottom edge of controls	

- 3 Select File > Save.

Resizing Controls




Grid Layout allows you to resize controls. If you resize a control in one mode, such as Edit, the control will also be resized in the layout for other modes, such as Base.

To resize controls

- 1 In the Web Layout Editor, hold down the Ctrl key, and then click the controls you want to resize.

The last item selected is the one to which all other items will be resized.

- 2 Perform the following tasks as necessary:

Task	Button
Make selected controls the same width.	
Make selected controls the same height.	
Make selected controls the same size.	

- 3 Select File > Save.

Spacing Controls




Grid Layout allows you to control the horizontal and vertical spacing of controls relative to each other.






To change the space between controls

- 1 In the Web Layout Editor, hold down the Ctrl key, and then click the controls whose spacing you want to modify.

The last item selected is the one that is used as the basis for the spacing.

- 2 Perform the following tasks as necessary:

Task	Button
Make horizontal spacing equal	
Increase horizontal spacing	
Decrease horizontal spacing	

Task	Button
Remove horizontal spacing	
Make vertical spacing equal	
Increase vertical spacing	
Decrease vertical spacing	
Remove vertical spacing	

- 3 Select File > Save.

Centering Controls



Grid Layout allows you to center controls vertically or horizontally.

To center controls

- 1 In the Web Layout Editor, hold down the Ctrl key, and then click the controls you want to center.

The last item selected is the one to which all other items will be centered.

- 2 Perform the following tasks as necessary:

Task	Button
Center controls vertically	
Center controls horizontally	

- 3 Select File > Save.

Setting Tab Order

Grid Layout allows you to set the sequence of controls that are activated each time a user presses the tab button. You can set the tab order by pointing and clicking in the Web Applet Layout Editor.

To set the tab order

- 1 In the Web Layout Editor, select Format > Set Tab Order from the main Tools menu.

The Web Layout Editor changes to Set Tab Order mode. Applet controls appear with a number next to them. The number indicates the sequence in which the user will progress through the controls using the tab button.

- 2 Define the tab order by clicking on controls in the sequence you want.

Each time you select a control the control is assigned a sequence number.

NOTE: If you do not click on a control, for example if you click on the layout background, the Web Applet Editor returns to normal edit mode.

- 3 Select File > Save.
- 4 Select Format > Set Tab Order.
The Web Layout Editor returns to normal edit mode.
- 5 Repeat steps [Step 1](#) through [Step 4](#) for each applet Web template mode.

Resizing the Grid Canvas

You can resize the grid canvas in the Web Layout Editor.

To resize the grid canvas

- 1 In the Web Applet Editor, use the scroll bars to display the bottom right corner of the grid canvas.

- 2 Place your cursor over the canvas border (bottom edge, right edge, or bottom right corner) and drag it to the new position.

The grid canvas is resized.

Setting a Default Method for an Applet

The default method is the one that is invoked when the user presses Ctrl + Enter. For applets in query mode, this is ExecuteQuery (pressing Alt + Enter will also execute the query). For other modes, the DefaultMethod applet user property can be set.

NOTE: This must be a valid applet InvokeMethod, such as NewRecord or GotoNextSet.

To set the default method for an applet

- 1 In Siebel Tools, select the Applet object, and then select the desired applet.
- 2 Expand the Applet object, and then select the Applet User Prop object.
- 3 Add a new record with the name DefaultMethod and the value of the method you want to be invoked.

About Views

Views define a visual representation of a business object's data.

A Siebel application window displays one view at any one time.

Views:

- Contain applets.
- Are components of a screen (each screen includes one or more views).
- Appear in one of the three primary view styles—list, detail, and explorer.

You create new views by using the View Wizard or by defining views in the view object in the Object List Editor.

NOTE: Views can appear in the user interface in the Show: combo box or as view tabs. See [“About The User Interface Navigational Paradigm” on page 504](#) for more information.

Creating Views Using the View Wizard

The View Wizard helps you create views and define all the necessary properties and child objects.

To create a view using the wizard

- 1** Check out and lock the project to which you want to add the view.
- 2** Choose File > New Object from the Siebel Tools main menu.

The New Object Wizard dialog box appears.

- 3** Click the View icon, and then click OK.
- 4** In the New View page, do the following:
 - Select the project that you want the view to be part of.
 - Enter a unique name for the new view.
 - Select the Business object whose data the view will display.
 - Enter the title for the view.
- 5** In the View Web Layout - Select Template page, select the template you wish to use for your new view.

For a complete list of View Web Templates, see *Siebel Developer's Reference*.

- 6** In the Web Layout - Applets page, select the applets that you want to appear in the Web layout and then click Next.

- 7 In the Finish page, review your selections, and then click Finish.

The View Web Template (Base) - Layout window appears. If necessary, you can drag and drop applets from the Applets Window to the sectors in the Web Layout Editor. For more information, see [“Editing the Web Layout of Views” on page 566](#).

- 8 To preview the view, right-click in the Web Layout Editor, and then choose Preview.
- 9 Save your new view by selecting File > Save.

Creating Views Using the Object Explorer

You can define views manually by using:

- Object List Editor, Property list window, or both
- Web Layout Editor

To define a view in the Object List Editor

- 1 From the Object Explorer, select View.

The Object List Editor opens.

- 2 Choose Edit > New Record to add a new view object definition.

- 3 Set the following properties for the new view record:

- **Name.** (Required.) The name of the view. All references to views are done through its name.
- **Business Object.** (Required.) The name of the business object used by the view. This determines the relationship between business components on which member applets are based.

All the sector properties must have a value specified (except explorer view).

- **Screen Menu.** If TRUE, the view should be included in the Site Map.
- **Title.** Text string used as the window title.

- **Visibility Applet.** A non-null value places the view in the Show: combo box. A null value places the view as a view tab.

See [“About The User Interface Navigational Paradigm” on page 504.](#)

- **Visibility Applet Type.** Limits view visibility by organization or responsibility. Valid values for Visibility Applet Type are:
 - All
 - Personal
 - Sales Rep
 - Manager
 - Organization
 - Sub-Organization
 - Catalog
 - Group

If Visibility Applet is blank, the view appears as a view tab and is not listed in the Show menu. Its visibility is determined by the value of Visibility Applet Type.

If Visibility Applet Type is blank, the view's visibility defaults to the most restrictive type defined for the business component. (This usually means Personal or Sales Rep visibility, depending on the corresponding visibility settings of the business component of the view's Thread Applet property.) The view is listed in the Show menu by the name given in the Visibility Applet property.

If both are blank, the visibility depends on the navigation context:

- When navigating from another view to this view using the Show menu, the query context and record focus are kept from the previous view, assuming that the business components and search specifications are relevant.

- When navigating to this view using the Site Map or the Screen tabs, the visibility defaults to the most restrictive type defined for the business component.

For more information on view visibility rules, see *Security Guide for Siebel eBusiness Applications*.

Editing the Web Layout of Views

You edit the Web layout of a view in the Web Layout Editor. The Web Layout Editor allows you to edit the mapping between applets in the view and placeholders in the template.

To use the Web Layout Editor to modify the layout of a view

- 1** Select a view in the Object List Editor.
- 2** Right-click, and then choose Edit Web Layout.

If the view has a template associated with it, the Web Layout Editor appears. The Web Layout Editor renders the underlying view template with mapped and unmapped placeholders.

- 3** Add an applet to the Web layout of the view by dragging it from the Applets window and dropping it onto an applet placeholder in the template.

The Applets window shows all applets based on business components in the business object of the view. When an applet is mapped to a placeholder, the applet is displayed in the position it would appear at runtime.

- 4** Delete an applet from the layout by selecting it, and then clicking the DELETE key.
- 5** Preview the view design by right-clicking on the layout, and then choosing Preview.

The preview simulates what the view would look like in the runtime environment by removing unmapped placeholders. This preview is not intended to be an exact representation of the eventual HTML output; it is intended to give you a rough idea about the structure and look of the generated output.

- 6 Export the preview to an HTML file by choosing File > Export, and then choosing a file name and location in the Save As dialog box that appears.
- 7 Change the Web template used by clicking the Change Template button that appears next to the Template text box in the Web Controls toolbar.

NOTE: This might result in some mappings becoming invalid if the corresponding placeholder IDs do not exist in the new template. You can test for invalid mapping by right-clicking and selecting Check Mapping.

For information about editing templates, see [Chapter 15, “Physical UI Navigation and Templates.”](#)

- 8 Save your changes by choosing File > Save.

Configuring Views for Personal Layout Control

Certain views in Siebel applications, such as home page views, allow the user to control the layout of the view. For example, these views allow the user to:

- Reorder applets
- Collapse or expand applets
- Show or hide applets

See the Customizing Your Home Page section in *Fundamentals* for a description of this feature from the end-user perspective.

The user can edit view layout in two modes. Show mode and Edit Layout mode.

- **Show mode.** Allows users to move, collapse and expand, show and hide individual applets within the view, using controls placed at the top of each applet on the view. For example, home page views are displayed in Show mode until a user clicks the Edit Layout button.

- **Edit Layout mode.** Allows users to move, expand and collapse, show and hide, individual applets, plus perform the following operations on all applets within the view: show all applets, hide all applets, minimize all applets, maximize all applets, and reset the default layout definition. This mode is presented to the user using a separate applet called the Layout Controls applet and appears after the user clicks the Edit Layout button.

To configure a view to support personal layout control

- 1** Set the User Layout property of the View's View Web Template object to TRUE.
- 2** Define the default layout of the view by setting the following properties of the View Web Template Items associated with the view.
 - **Display Size.** Determines whether the applet is minimized or maximized. Always Maximized indicates that the applet cannot be minimized by the end user.

NOTE: Minimized and maximized are referred to as collapse and expand in the client user interface.

- **Display Visibility.** Determines whether the applet is shown or hidden. Always Show indicates that the applet cannot be hidden by the end user.
- **Move Range.** Defines a range in which the applet may be moved. For example, on an application home page with two columns, applets would specify a move range of either *Column1* or *Column2*. Any applet with a move range of *Column1* is movable only within the first (left) column. Any applet with a move range of *Column2* is movable only within the second (right) column.

If this property is not defined, the applet cannot be moved by the end-user. The applet location is fixed within the view. For example, the salutation applet on the home page would typically not have move range specified for it.

- 3** Add the Layout Controls Applet to the view, add a corresponding View Web Template Item, and map the applet to a placeholder in the Web Template.

There is an applet in the Siebel repository called Layout Controls applet. Add this applet to the view that you want to enable for personal layout control. It serves as a container for the controls that handle view-level operations, such as Reset Default Layout. In Show Mode, this applet appears as the Edit Layout button. In Edit Layout mode, which appears after the user clicks the Edit Layout button, this applet shows all applets on the view, and allows the user to Hide All Applets, Show All Applets, Reset Default Layout, and return to Show mode by clicking Done.

- 4** Add the following view layout controls to applets within personalized views, add corresponding Applet Web Template Items, and map the controls to the appropriate placeholders in the Web template.

- ButtonMoveAppletUp
- ButtonMoveAppletDown
- ButtonHideApplet
- ButtonShowApplet
- ButtonMinimizeApplet
- ButtonMaximizeApplet

These view layout controls use Invoke Methods to manipulate the user's view layout preferences.

Providing User Access to a New View

After creating a new view, you need to register the view in the Siebel application. This will provide users access to a new view. This is an application administration task.

NOTE: This is an important step to remember. If you simply define a view without providing access to it, it will not be accessible to any users in the Siebel application client. You also need to do this to be able to test the new view in the Web client.

Once you add a new view to the Siebel repository, an application administrator needs to do the following:

- Log in to the Siebel application as a user with the right level of administrative responsibility.
- Add the new view to the list of views.
- Add the view to the appropriate responsibility.

Depending on the nature of the new view and the users who need access to it, the application administrator may also need to:

- Add new responsibilities.
- Add employees to the new responsibilities.

For more information about doing these tasks, see *Applications Administration Guide*.

Reasons a View Is Not Visible to a User

When a view is not visible to the logged-in user, there are the following possible reasons:

- The view does not exist in the .srf file.

This includes a possible misspelling when the view was registered (Site Map > Application Administration > Views); that is, it does not match the view name in the .srf file. If it matches, compile the .srf file again using the All Projects option (full compile).

- The view is not included in one of the logged-in user's responsibilities.

- Determine which responsibilities the logged-in user has (Site Map > User Administration > Employees).
- Determine for each responsibility whether the view is included (Site Map > Application Administration > Responsibilities).

- The view is hidden using personalization rules.

Determine this under Personalization Administration > Views. For testing purposes, you can also switch off the EnablePersonalization parameter in the .cfg file.

- The view is not included either in the menu or in the view tabs. In this case, the view can only be accessed by drilling down from another view.

- In Siebel Tools, examine the Screen Menu property of the View object. It must be set to TRUE for the view to be included in the Site Map.
- Determine whether the view is included in a screen and that the Viewbar Text property of the Screen View child object of the screen is set appropriately.
- Determine whether the view's Visibility Applet and Visibility Applet Type properties are set correctly. For more information, see [“Creating Views Using the Object Explorer” on page 564](#).

- The view belongs to a screen that is not included in the currently running application.

- In Siebel Tools, determine whether the screen is included in the application (Screen Menu Item child object of the application).
- Determine whether the application name is spelled correctly in the .cfg file.

- The view does not belong to the same business object as the screen's default view.

Make sure that the view is based on the same business object.

For restrictions on the Screen property, see *Object Types Reference*.

- The view is not available due to upgrade problems.

If an upgrade was done, make sure that it was successful by verifying all the log files that were created. The upgrade log files are found in the `DBSERVER_ROOT\DB_PLATFORM` directory.

- The view is not included in your license keys.

If none of the previous reasons is responsible for the view not being visible, it is likely that the view is not included in your license keys. Send the license keys to Siebel Expert Services for examination. See also Alert 0041 on Siebel SupportWeb.

- The screen menu item or page tab is not translated into its target language.

Make sure that for each screen associated with the application (Screen Menu Item object) there is a translated string available in the target language and a Screen Menu Item Locale child object. If not, the screen will not appear in the Site Map.

Similarly, for a page tab to appear, the Page Tab object must have a translated string and a Page Tab Locale child object with the appropriate language code.

For example, if the application runs in Norwegian, there must be Screen Menu Item Locale and Page Tab Locale objects with the Language Code property set to NOR.

About Screens

A *screen* is a collection or group of related views:

- The screen represents a logical grouping of views pertaining to one business function.

- All the views in a screen usually map to a single business object.

You access screens through the Site Map or the tabs in the Tab bar. The menu and tab that provide access to a screen are defined as part of an application object definition—one or both may exist for a given screen. Screen definitions specify the default view that appears when a tab is clicked.

Screens have a single child object type: Screen View. The screen view object type controls which views appear in the fourth-level navigation Show: combo box.

NOTE: The Site Map is limited to visibility-level views only. Non-visibility level views, such as Account Profile View and Account Attachment View, do not show up on the Site Map.

See [“About The User Interface Navigational Paradigm”](#) on page 504.

Defining Screens

You create screens by creating new objects and defining object properties in Siebel Tools. There is no wizard to walk you through the process.

To define a screen

- 1 From the Object Explorer, select Screen.
The Object List Editor opens.
- 2 Choose Edit > New Record to add a new Screen object definition.
- 3 Set the following properties for the new Screen record:
 - **Name.** (Required.) Name of the screen. All references to a screen are done through its name.

- Default View.** View that will be used when the user clicks on a page tab for a Screen.

NOTE: A view must be added to the screen before it can be specified as a default.

Figure 161 shows the definition for the Accounts Screen.

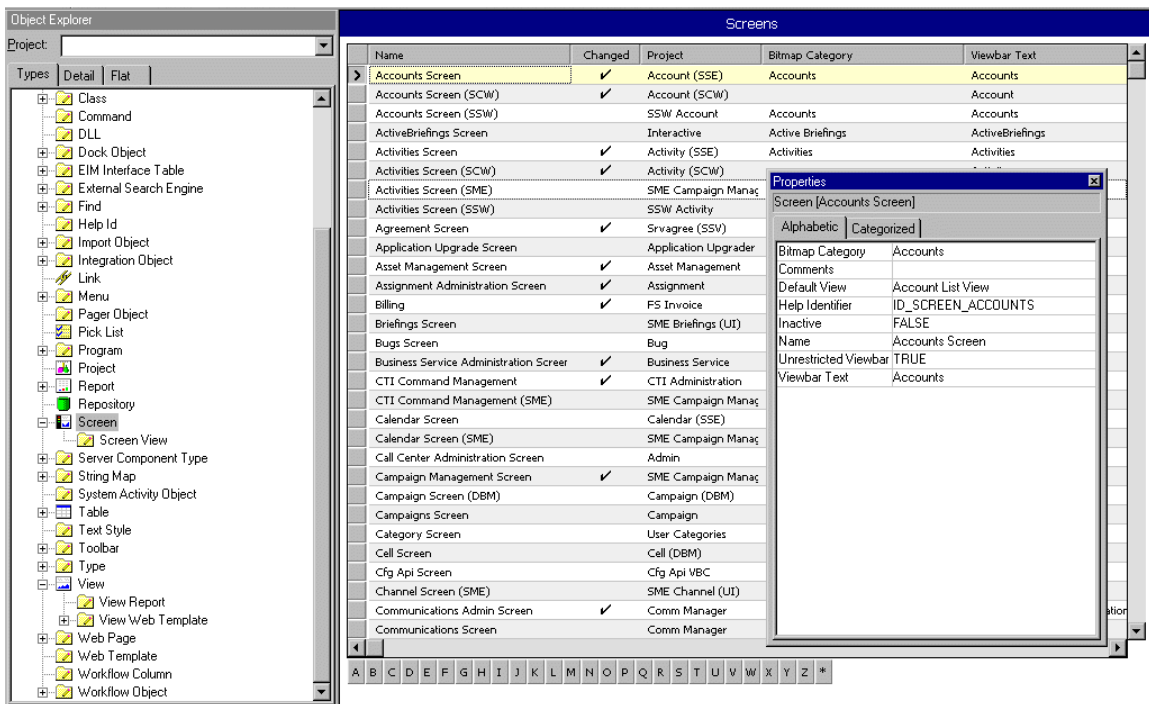


Figure 161. Accounts Screen Definition

To define a screen view

- From the Object Explorer, select the Screen object type. The Object List Editor opens.
- Choose Edit > New Record to add a new screen object definition.

3 Set the following properties for the new Screen View object:

- **Menu Text.** Both the Screen View Menu Text property and the Screen Menu Item display value show up in the Site Map. The former shows up as a subitem under the latter.
- **Sequence.** Specifies the order in which the views will appear in the cascading menu for the Screen menu item.

If you want the view to appear in the View Category Show: combo box, set the following properties. Typically, Category Menu Text and Category Viewbar Text are the same value.

- **Category.** This is a logical representation of what category the view belongs to.
- **Category Menu Text.** The display value that appears within the combo box.
- **Category Viewbar Text.** The last is the display text of the View tab under which it appears.

See [“About The User Interface Navigational Paradigm” on page 504](#) for more information.

Figure 162 shows the Account Address view definition in the Object List Editor and Properties windows.

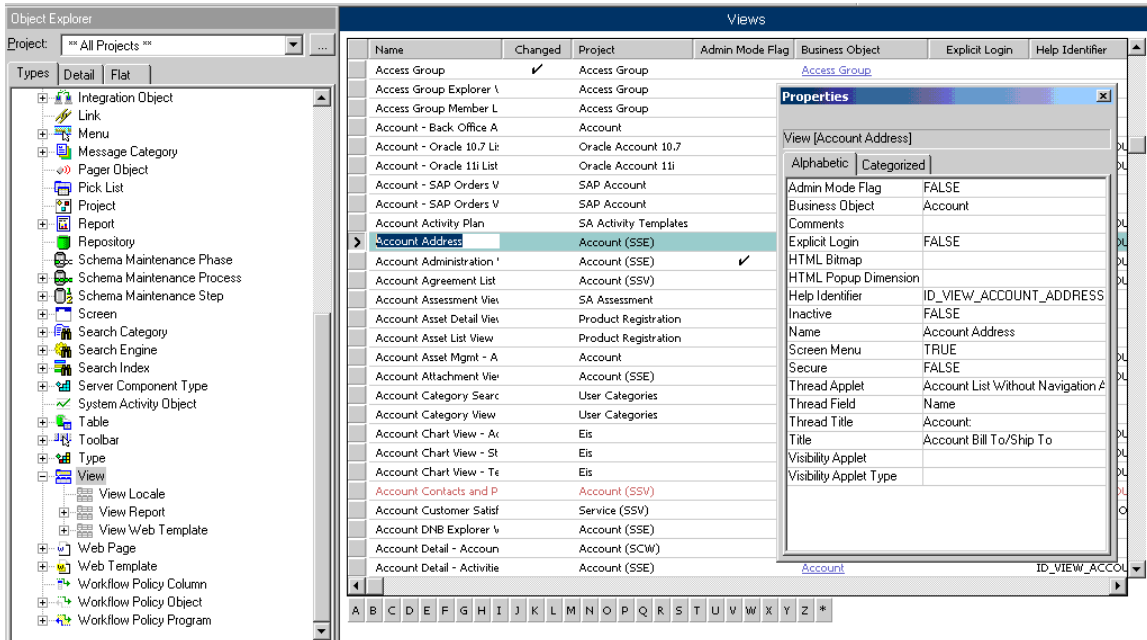


Figure 162. Account Address View Definition

The Applets window shows only the available applets in either icon or list form. An applet is considered available if it is associated with a business component that is present in the business object for this view.

You can delete an applet on the layout by selecting it and pressing the DELETE key.

About Applications

An *application* is a collection of screens—it defines which screens will be accessible through menus and tabs. You can create new applications, but developers should rarely do so. Typically, developers will modify existing applications to meet an organization’s requirements.

Many applications can be supported in a single repository.

The application's name is the one used in the configuration file that is read when the application is executed. Application names are case-sensitive and space-sensitive.

Siebel applications place application object definitions in their own separate project. This is done to minimize locking of the application object definition.

If you are adding new screens to an application you need to:

- Associate screens with Page Tabs
- Associate screens with Screen Menu Items

Associating Screens with Page Tabs

Page Tabs are a child object type of the Application object type. They appear as the first-level navigation in the user interface, allowing users to click the tab to go to the associated screen. [Figure 163](#) shows Page Tabs in the user interface.



Figure 163. Page Tabs in the User Interface

To create *Page Tabs* in an application

- 1 In the Object Explorer, expand the Application object type.
- 2 Select the child object of Application called Page Tab. The Page Tabs List Editor appears as the bottom windowpane in the Object List Editor window.
- 3 With the Page Tabs List Editor active, choose Edit > New Record to add a new Page Tab object definition.
- 4 Set the properties for the new Page Tab record:
 - **Screen.** The screen you want to expose through a page tab.

- **Sequence.** Specifies the order of the page tabs for an application.
- **Text.** Specifies the text that will be displayed on the page tab.

Defining Screen Menu Items

Screen Menu Items are child objects of the Application Object Type. Screen Menu Items appear as hyperlinks on the Site Map. They allow users to click the hyperlink to go to the screen. [Figure 164](#) shows Screen Menu Item definitions displayed on the Site Map.

NOTE: The site map is available by choosing View from the application-level menu and then choosing Site Map.



Figure 164. Screen Menu Items Displayed on the Site Map

To associate screens with screen menu items

- 1 From the Object Explorer, expand the Application object type.
- 2 Select the child object called Screen Menu Item.

The Screen Menu Items List Editor appears as the bottom windowpane in the Object List Editor window.

- 3 With the Screen Menu Items List Editor active, choose Edit > New Record to add a new Screen Menu Item object definition.
- 4 Set the properties for the new Screen Menu Item record:
 - **Screen.** The screen that will be accessed with the menu item.
 - **Sequence.** The position of the menu item on the Site Map for the application.

Duplicate sequence numbers are acceptable—in that case the first one added will be the first to appear.

You can also use gaps in sequence numbers so that you can insert additional screens between your original entries.
 - **Text.** The text to use for the menu item.

About Web Pages

The Web Page Object is the top-level object in the Web hierarchy that is used to create Web pages such as the following:

- Login pages
- Error pages
- Container pages

Editing the Layout of Web Page Objects

Like applets and views, Web pages are associated with templates. Web Page Items (child objects of Web pages) are mapped to placeholders in the templates. The Visual Web Page Editor allows a user to view and edit Web page objects.

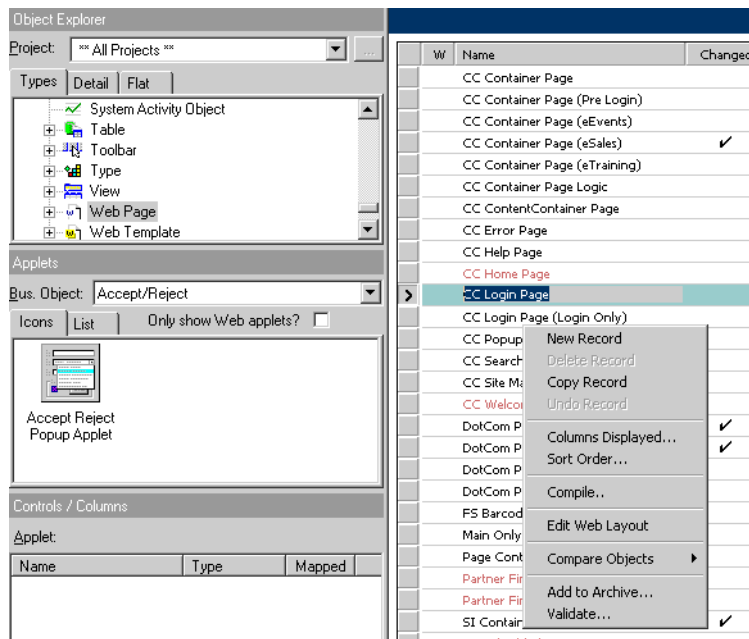
To access the Web Page Layout Editor

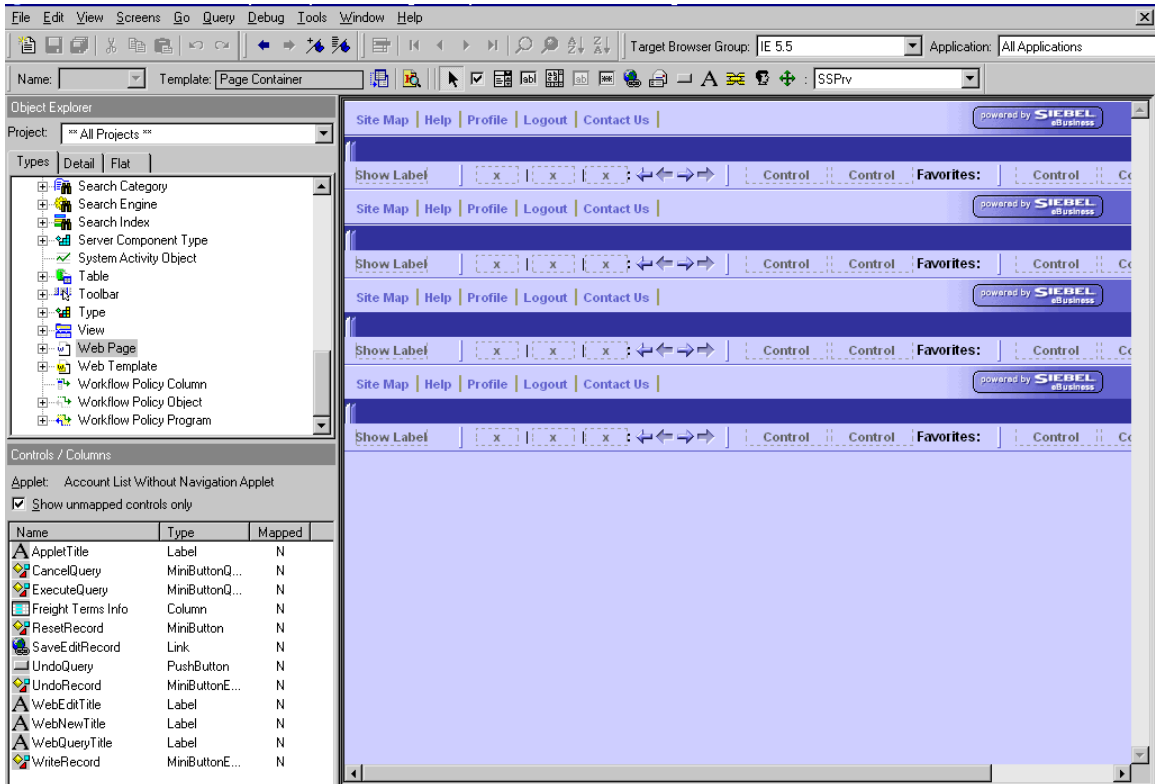
- 1 Select a Target Browser from the drop-down list on the toolbar.

NOTE: If you do not select a browser, an error message appears when you choose the Edit Web Layout option from the menu.

- 2 Lock your project.
- 3 In the Object Explorer, select the Web Page Object.
- 4 In the Web Pages list object editor, select the Web Page you want to modify, right-click, and then choose Edit Web Layout.

The Web Page-Layout (Container Page) window appears.





NOTE: The layout editor shows multiple images because the template on which the Web page is based contains a conditional tag such as `< swe:if >` and `< swe:case >`. The template content that is used varies depending on whether any one of the conditions is met. For example, in the case of the Page Container, the condition can simply be whether or not a CTI Java Applet is used or some other subtle or nonvisual differences. The effect is that the layout editor shows the page as if all the conditions were true. This is useful in case you want to edit any of them. However, typically only one condition would be true at run time, so you would not see the repeating images in the Web client.

- 5 Select a custom control from the combo box on the toolbar and drag it to a place holder.

- 6 Set properties (like caption, invoke method, and so on) for the control using the Properties window.

If you click on the Web Page Item object type in the Object Explorer, the Web Page Items list applet displays these mappings.

Associating Images With Siebel Objects

You can associate image files, such as GIF and JPG, with Siebel objects, including:

- Toolbar buttons
- Command buttons
- Icon maps
- Page tabs
- Tree nodes
- Views

Images are defined in the repository using the Bitmap object type. Image files can be of any format supported by the target browser. The Bitmap object type identifies the location of the image file and other properties, such as width and height. See *Object Types Reference* for more information about the properties of the Bitmap object type.

NOTE: Only images that are associated with Siebel objects, such as icon maps, page tabs, and so on, are defined as Bitmap objects in the Siebel repository. Some images used in Web templates, such as static images, are not associated with Siebel objects and are not defined as Bitmap objects in the Siebel repository. These objects are defined in the application's configuration (.cfg) file.

The Bitmap Category object type allows you to group image files together by function. For example, there is a category called Button Icons that contains all images for buttons on applets used in Siebel applications.

Figure 165 shows bitmaps associated with the Calendar bitmap category.

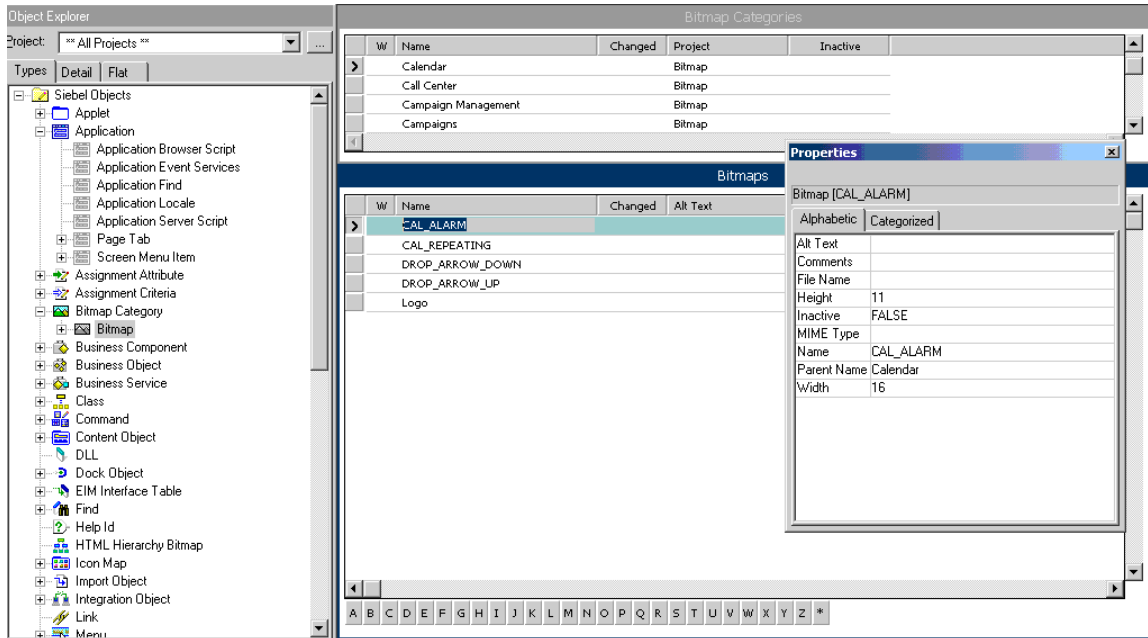


Figure 165. Bitmaps Associated with the Calendar Bitmap Category

To create a new Bitmap Category and associate bitmap objects to it

- 1 In the Object Explorer, select Bitmap Category.
- 2 Add a new bitmap category.
- 3 Select the Bitmap child object.
- 4 Add a new bitmap record.
- 5 Enter the name of the image file you want to use.

NOTE: Image files are stored in the following directory:
Siebel_installatoin\PUBLIC\language_code\IMAGES

- 6 Preview the image to verify that it is the correct one and the correct size.

Using Bitmap Objects with Button Controls

You can associate bitmap objects with button controls to display images instead of text, much like a Toolbar icon. Unlike a Toolbar icon, however, a bitmap button control is a command button in the applet. A good example of the use of a Bitmap object with a button control in an applet is the More/Less button. The More/Less button appears in the upper-right corner of many applets. The control uses a bitmap object called BTTNS_MORE that belongs to the Bitmap Category HTML Control Icons.

You associate Bitmap Object with button controls using the following properties of the Button Control object definition:

- HTML Bitmap. Defines the Bitmap object used when the button is active.
- HTML Disable Bitmap. Defines the Bitmap object to use when the button is inactive.

See [“Image Support” on page 891](#) for more information about how images are handled by Web templates.

Defining Toolbars and Menus

You can create new toolbars for an application by defining a toolbar object and modifying the appropriate Siebel Web templates to expose the new object in the user interface. You can also add new icons to existing toolbars.

To define a new toolbar

- 1 In Types view in the Object Explorer, double-click the Toolbar object type.
- 2 Click to the left of a row in the Object List Editor, and then choose Edit > New Record.
- 3 Specify the name of the new toolbar in the Name property of the new object definition.

- 4 To expose the toolbar to the user interface, you need to add a specific tag to the Container Page or one of its child templates that you are using.

For detailed information on templates and tags, see *Siebel Developer's Reference*.

To add a new toolbar icon to an existing toolbar

- 1 Verify that the bitmap image you want to use on the toolbar icon surface currently exists as a child bitmap object definition of the bitmap category object definition named Command Icons.

If it does not exist, create a bitmap object definition in this bitmap category as described in [“Associating Images With Siebel Objects”](#) on page 582. If it does exist, note the name of the bitmap object definition.

- 2 Verify that the method you want for this toolbar icon to invoke currently exists, or add a Siebel VB or eScript script to the application PreInvokeMethod.

You need to write an If or Case statement based on MethodName and write the instructions for that MethodName within the If or Case statement.

You also need to change the last line of PreInvokeMethod to CancelOperation (from ContinueOperation).

- 3 Navigate to the Command object type in the Object Explorer, and add a new Command object definition in the Object List Editor. Specify the HTML bitmap to use in the Bitmap property, the method to invoke in the Method property, and other properties.
- 4 Navigate to the Toolbar object definition to which the new Toolbar Item is to be added.
- 5 In the Object Explorer, select the Toolbar Item object type.
- 6 In the Object List Editor, add a new Toolbar Item object definition. Specify the name, the name of the Command object definition which supplies the bitmap and method, and the sequence number of the toolbar icon relative to the other toolbar icons appearing in the Toolbar Items list in the Object List Editor.

Creating Command Objects

A command object definition specifies which invoke method is called when a toolbar icon, application-level menu item, or applet menu item is associated with the command is executed. It also specifies which bitmap appears on the toolbar icon for the toolbar items. Command object definitions are referenced by Toolbar Item, Menu Item, or Applet Method Menu Item object definitions.

NOTE: If you look in the target property for the command object, you will see six values in the picklist. The only valid values are Browser and Server. These values are the only options when you use the Command Wizard to create a new command object.

Using the Command Object Wizard

The Command Object wizard creates Command object definitions by helping you select the appropriate properties for the object.

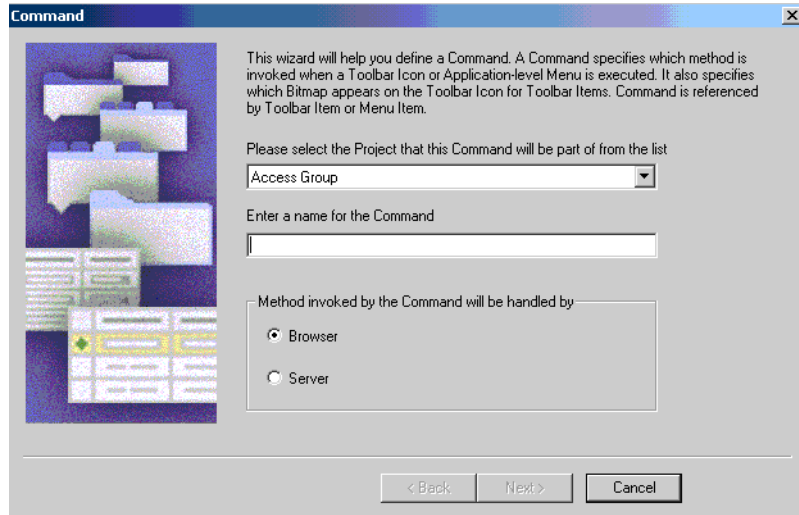
To use the Command Object Wizard

- 1 Choose File > New Object.

The New Object dialog box appears.

- 2 Click the Command object icon.

The Command dialog box appears.



- 3 In the Command dialog box, do the following:
 - Enter the project for which you wish to create the new command object.
 - Enter a unique name for the command object.
 - Select whether you want the Method invoked by the command to be handled by the browser or the server.
 - Click Next.
- 4 In the next dialog box, do the following:
 - Select the object that will handle the command. If the command is to be handled by a business service, specify the business service from the drop-down list. You must know whether the selected business service is available for your choice of browser/server.

- Enter the Method to be invoked by the command. Specify the method to be invoked by the command. You are responsible for choosing a method that is available for the business service or application chosen.
- You can provide the argument to be passed to the method (this is optional). The argument must be correct for the chosen method.
- Click Next.

Command

Select the object that you want to handle this Command.

Application

Business Service:

Command to execute

Enter the method and argument to be invoked with this Command.

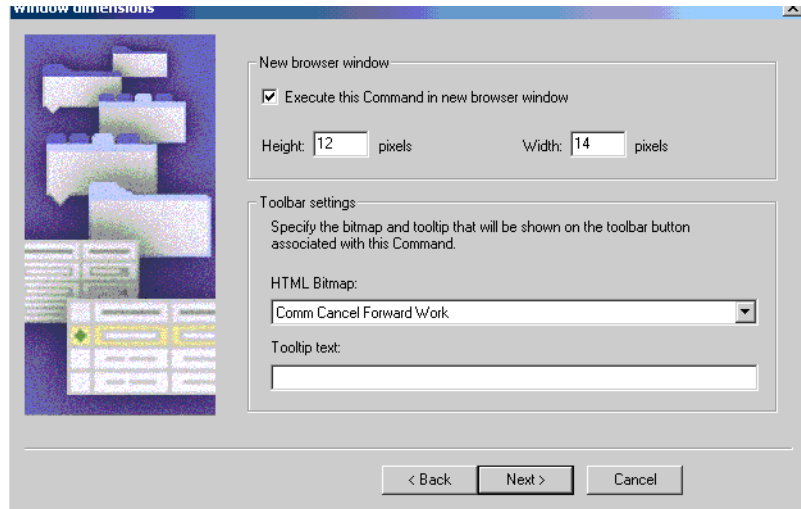
Method to be invoked:

Argument for the method:

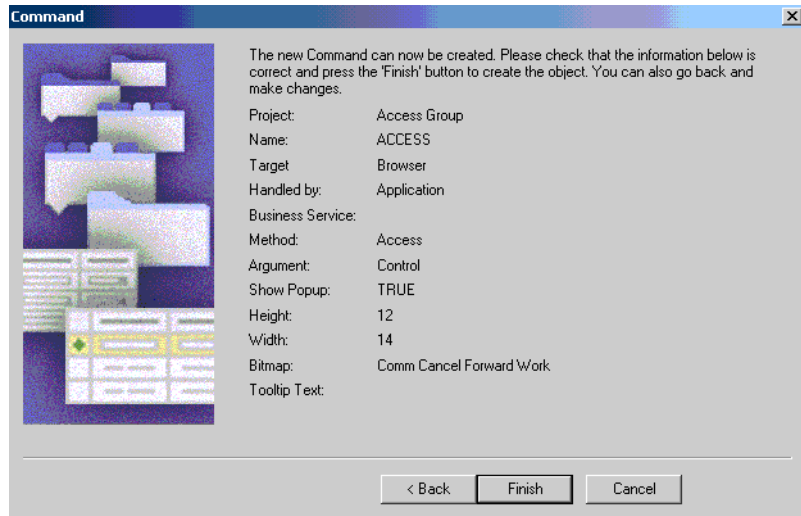
< Back Next > Cancel

- 5 In the Window dimensions dialog box, do the following:
 - Specify if the command should be executed in a new browser window. If it should be, the window's height and width must be specified. Both these values must be valid integers.
 - Specify the HTML bitmap and the tooltip text to be shown on the toolbar button associated with the command (optional).

- Click Next.



- 6 In the Command dialog box that appears review your entries.



- 7 If any corrections need to be made, click Back to return to the appropriate page on which the correction is to be made.

If the properties are correct, click Finish to generate the command object. After the object has been generated, the user will be taken to its definition in tools.

Using JavaScript to Extend Toolbars

You can take advantage of the functionality of high interactivity by extending JavaScript toolbars and creating new ones. Creating new JavaScript files to extend JavaScript toolbars is necessary if you need more toolbar icon types than the standard ones (Link, Button, Label, and Separator in Siebel 7.5).

To extend a toolbar using JavaScript

- 1 Create a JavaScript file to define an extended JavaScript toolbar class that is a subclass of JSSToolbar.
- 2 Copy the JavaScript file to
`SIEBSRVR_ROOT\webmaster\<Siebel_build_number_in_use>\scripts.`
- 3 In Siebel Tools, create a DLL object as shown in the following table.

Field	Value
Name	User-defined name for the DLL object, for example BarcodeToolbar
Project	A currently locked project in the Siebel Repository
File Name	File name that references the JavaScript file, for example barcodeToolbar.js

- 4 Create a Class object as shown in the following table.

Field	Value
Name	Name of the class defined in the JavaScript file, for example JSSBarcodeToolbar
Project	The locked project used in Step 3

Field	Value
DLL	Name of the DLL object created in Step 3
High Interactivity Enabled	1

- 5 If you are creating a new toolbar, create a Toolbar object, as shown in “[To define a new toolbar](#)” on page 584.

The Class property must be the class defined in the JavaScript file, for example JSSBarcodeToolbar.

- 6 Add new toolbar items as shown in “[To add a new toolbar icon to an existing toolbar](#)” on page 585.

- 7 If you are creating a new toolbar, add a `< swe: toolbar >` tag to the appropriate Web template as shown in “[HTML and JavaScript Toolbars](#)” on page 866.

The name property in the `swe: toolbar` tag must be the name of the Toolbar object in [Step 5](#).

- 8 Add `< swe: toolbaritem >` tags to the appropriate `swe: toolbar` tag as shown in “[HTML and JavaScript Toolbars](#)” on page 866.

Creating Applet Menus

You can configure the applet menus that come with Siebel applications and also create custom applet menus of your own using the Applet Method Menu Wizard.

Defining Web Menus Using the Applet Method Menu Wizard

The Applet Menu Wizard will allow the user to modify an applet's method menu (applet level menus). Applet method menus are constructed by inheriting method menu items from the class to which the applet belongs and its super classes, and also by explicitly creating method menu items for the applet. Using the wizard, you can do the following:

- Suppress inherited method menu items

- Resurrect inherited method menu items
- Create new method menu items for an applet
- Delete existing method menu items of an applet

To use the Applet Method Menu Wizard

- 1** Choose File > New Object.

The New Object dialog box appears.

- 2** Select the Applet Method Menu object.

The Applet Method Menu dialog box appears.

- 3** In the Applet Method Menu dialog box, do the following:

- Specify the applet whose method menu you wish to modify and also the project for the applet.

Only projects locked by the user will appear in the drop-down list.

- Click Next.

- 4** In the second Applet Method Menu dialog box, do the following:

- Make a menu item visible in an applet by moving the item to the Selected Menu Items window.

- Suppress a menu item by moving it out of the Selected Menu Items Window.

- Click Finish or select Create New Menu Item. If you click Finish, all the changes that you have made are committed to the repository, and you are taken to the applet's object definition in the Object List Editor. If you select Create New Menu Item, the Finish button becomes the Next button.

- 5** If you are creating a new menu item, and you have checked the box labeled Create New Menu Item on the Applet Method Menu dialog box, then click Next.

- 6** Create a new method menu item definition by selecting an entry from the Select the Command to be executed by this Menu field.

- 7** From the Enter the text to be displayed for this Menu Item field, specify the text to display for this method menu item, and then click Next.

The Applet Method Menu Item dialog box appears.

You can examine the properties that you specified earlier. If any changes have to be made, you can click Back and return to the appropriate dialog box on which the correction is to be made.

- 8** If the properties are correct, you can click Create Menu Item to generate the method menu item object. After the item is generated, the Back button is deactivated and the Next button becomes enabled.

- 9** Click Next.

The second Applet Method Menu dialog box appears and the method menu item you just created is displayed in the Selected Items list box.

- 10** Click Finish.

The Applet Layout screen appears.

Creating Menus Manually

You can also create menus by creating an Applet Method Menu Item manually in the Object List Editor.

To use the Applet Method Menu Item Object to create the menu manually

- 1** Choose Applet > Applet Method Menu Item object.
- 2** Select an entry in the Applets object list editor.
- 3** Right-click and select Edit Web Menus.

The applet-menu window appears for you to create your menus.

Configuring Keyboard Accelerators

Keyboard accelerators are implemented using the command architecture. They are configured in Siebel Tools using the Accelerator object, which is a child of the Command object. Since accelerators are mapped directly to commands, the scope of the actions represented may be specific to the currently active applet, or it may apply to the application session as a whole. For example, an accelerator to initiate a new query will have specific focus on the current applet, while an accelerator to invoke the Site Map page is independent of the current application context.

Commands must be loaded into the active menu structure at runtime in order to be available. This means that the command represented by each accelerator must be available to the user at a given point in time for the associated keyboard accelerator to be active. For a command to be available to the user, it must be associated with either the application menu or the applet-level menu for the currently active applet.

Adding a New Keyboard Accelerator

You can add new keyboard accelerators.

To add an accelerator

- 1** Make sure the action to be performed by the accelerator is represented by a command object in the Siebel Repository.

If not, you must add a command object. See [“Creating Command Objects” on page 586](#) for information.
- 2** Make sure the command to be mapped to the new accelerator is included as part of the active menu hierarchy, at either the application or applet level, for the application contexts in which the accelerator will be active.
- 3** Navigate to the Commands object in the Object Explorer view.
- 4** Select the command that you want to modify.
- 5** Expand the Command object type, and then select the Accelerator child object.
- 6** Add a new record.
- 7** Specify the key sequence.

- 8 Specify the display name to be associated with the accelerator.
- 9 In the Platform field, specify the keyboard enablement mode or modes for which this accelerator will be active.
 - Extended for extended mode only
 - Basic for basic mode only
 - All for both modes
- 10 Compile and check-in the project.

NOTE: Keyboard accelerators for commands related to the Siebel Communications Server are configured through administrative screens in the application. They are not compiled in the Siebel Repository File using Siebel Tools. Any such accelerators defined through the Siebel Communications Server administrative screens will take precedence over accelerators defined in the Siebel Repository File for identical key sequences. For more information, see *Siebel Communications Server Administration Guide*

Modifying the Key Sequence for an Existing Accelerator

You can modify the key sequence for an existing accelerator.

To modify the Key Sequence for an accelerator

- 1 Navigate to the Commands object in the Object Explorer view.
- 2 Identify and highlight the command to modify.
- 3 Select the accelerator that you want to modify (Accelerator is a child object of the Command object).
- 4 Modify the Key Sequence property for the accelerator.

See [“Design Considerations for Keyboard Accelerators”](#) on page 596.
- 5 Compile and check-in the project.

Hiding The Key Sequence in the User Interface

You can hide the key sequence so that it is not displayed in the user interface. The Key sequence for a given accelerator is defined by the Display Name property of the Accelerator Locale record. To hide the key sequence, leave this property blank.

Design Considerations for Keyboard Accelerators

Key sequences that resemble the underlying command are useful since they help the user remember the accelerator (for example, “Ctrl + N” for “New Record”). However, it can be beneficial to keep several design considerations in mind to help maximize productivity gains when adding or modifying keyboard accelerators.

For applications running extended mode keyboard enablement, take care not to override native browser functionality that the user community uses regularly. For example, “Ctrl-C” copies a text string to the clipboard in Microsoft Internet Explorer 5.5. This could be a useful feature within the Siebel application browser environment for managing text.

Keep related accelerators grouped as much as possible in terms of mapped key sequences. For example, mapping key sequences that all start with “Ctrl + Alt...” for query management functions can assist users in remembering related accelerators.

Take care not to map frequently used commands to key sequences that are similar to those of commands with severe results. For example, mapping a frequently used command to “Ctrl + Alt + X” may lead to unwanted accidental logouts for users if the “Logout” command is mapped to “Ctrl + Shift + X”.

Keyboard accelerators for commands related to the Siebel Communications Server are configured through administrative screens in the application. They are not compiled in the Siebel Repository File using Siebel Tools. Any such accelerators defined through the Siebel Communications Server administrative screens will take precedence over accelerators defined in the Siebel Repository File for identical key sequences.

For more information about Keyboard Accelerators, see *Applications Administration Guide*.

Configuring Spell Check

Siebel Spell Check is available for use by many Siebel eBusiness applications. This feature helps users, such as call center agents, identify spelling errors before sending out emails, faxes, or other text communications. For additional information on Siebel Spell Check, see *Siebel eMail Response Administration Guide* and *Siebel Call Center User Guide*.

Users can invoke Siebel Spell Check from an applet-level menu item. You configure this menu item by creating a “Check Spelling Field” user property for the applet where the Check Spelling button and the field to be checked are located. The workflow is described in the sections that follow.

- 1 Create a Spell Check button.** You need to create a Spell Check button for the applet containing the field to be checked. For information on performing this procedure, see [“To create a Spell Check button.”](#)
- 2 Set the Spell Check button user properties.** For information on performing this procedure, see [“To set the Spell Check button User Properties” on page 598.](#)
- 3 Edit the Web Applet template.** After creating a Spell Check button, you add it to the template where it will appear. For information on performing this procedure, see [“To edit the Web Applet” on page 599.](#)
- 4 Add the Spell Check business component to a business object.** Next, you add the Spell Check business component to the business object of the applet containing the field to be checked. For information on performing this procedure, see [“To add the Spell Check business component to a business object” on page 599.](#)
- 5 Create a Spell Check menu item.** You need to create a Spell Check menu item for the applet containing the field you want to be checked. For information on performing this procedure, see [“To create a Spell Check Menu Item” on page 600.](#)
- 6 Compile the project.** For information on performing this procedure, see [“Compiling Projects” on page 130.](#)
- 7 Test the applet.** You can test the applet by running a Siebel application, navigating to the applet you have configured, and making sure that the Check Spelling button is present and working correctly.

To create a Spell Check button

- 1** In the Object Explorer, double-click the Applet object type to expand it.
- 2** In the Applets window, select the name of the applet for which you are creating a Spell Check button.
- 3** In the Object Explorer, select the Control object type and add a new record.

Create the new record with the following values:

Field	Value
Name	ButtonCheckSpelling
Caption	Check Spelling
Field	[field name]
HTML Type	MiniButton
HTML Only	True
Method Invoked	ShowPopup

NOTE: If the Method Invoked value does not appear in the pick list, type it in.

To set the Spell Check button User Properties

- 1** In the Object Explorer, double-click the Controls object type to expand it.
- 2** Select the Control User Properties object type.
- 3** Click Control User Properties.

- 4 In the Control User Properties window, create new records for the following user properties, with the corresponding values:

Field	Value
Mode	Edit
Popup	Spell Checker Popup Applet
Popup Dimensions	560 X 350 (recommended initial size)

To edit the Web Applet

- 1 In the Object Explorer, double-click the Applet object type to expand it.
- 2 In the Applets window, select the name of the applet for which you are creating a Spell Check button, and then right-click and choose Edit Web Layout.
- 3 In the Web Control toolbar, from the Mode drop-down list, select Edit.
- 4 In the Controls window, select the “Check Spelling” icon, and then drag it to a placeholder in the Web template.
- 5 In the Web template, right-click and choose Preview.

You can see the Spell Check button as it will appear in the user interface.

To add the Spell Check business component to a business object

- 1 In the Object Explorer, click the Business Object type.
- 2 In the Business Objects window, select the business object to which you want to add the Spell Check business component.
- 3 In the Object Explorer, double-click Business Object to expand it, and then select Business Object Component.

- 4 In the Business Object Component (child) window, add a new record.

Create the new record with the following values:

Field	Value
BusComp	Spell Checker Applet VBC

To create a Spell Check Menu Item

- 1 In the Object Explorer, click Applet, and then select the applet for which you want to create a Spell Check menu item.
- 2 Double-click the Applet object type to expand it, and then select Applet Method Menu Item.
- 3 In the Applet Method Menu Item window, add a new record.

Create the new record with the following values:

Field	Value
Command	Check Spelling
Menu Text	&Check Spelling
Position	2

- 4 Select the Applet User Property object type.
- 5 In the Applet User Properties window, add a new record.

Create the new record with the following values:

Field	Value
Name	Check Spelling
Value	[Name of the field that will use Spell Check]

Using the Locale Management Utility

The Locale Management Utility (LMU) in Siebel Tools helps you manage the process of localizing text strings, such as field labels, and other locale-specific attributes, such as the height and width of controls. This includes exporting the strings to a file, which is then translated and imported back into the repository. The LMU provides the export and import tools to do this.

You use the Locale Management Utility to:

- Find strings that need to be translated.
- Find existing translations to use for untranslated strings.
- Export strings and locale-specific attributes to a file for localization.
- Import strings and locale-specific attributes from a file back into the repository.
- Search for strings and locale-specific attributes that have changed since the last export.
- Compare object definitions in the repository to the object definitions stored in the export file.

Finding Untranslated Text Strings

You can use the Locale Management Utility to find text strings in the repository that have not been translated or need to be re-translated since the source string has changed since the last translation.

NOTE: The LMU performs search and comparison functions at the object level, not the attribute level. Therefore, if a locale object contains multiple string attributes, the search function will return all strings contained in the locale object, even if only one of them has been translated.

To find and export strings that need to be translated

- 1** Choose Tools > Utilities > Locale Management.

The Locale Management Utility appears.

- 2** Select the source language and the target language.
- 3** Select the applications or projects that you want to localize.
- 4** Select the Untranslated Strings tab.
- 5** If you also want to see strings that have been marked as Redo, select the *Report string attributes of objects marked with 'Redo' flag* check box.

The Redo flag is marked when a record in the repository has been changed since the last time export occurred and therefore may need to be translated again.

See [“Identifying Modified Objects” on page 608](#) for more information about Redo.

- 6** Click Find Strings.

The Locale Management Utility searches through the string attributes of objects in the selected applications or projects and displays the ones that have not been translated and, if *Report string attributes of objects marked with 'Redo' flag* check box was selected, the strings that need to be re-translated are also displayed.

- 7** After you find untranslated strings you can perform the following tasks:

- Find the views that the untranslated strings belong to by clicking the Find View button.
- Go to the parent object of the string in the Object Explorer by selecting a string, and then clicking the Go To button.
- Export all untranslated strings to a text file by clicking the Export button.

Finding Existing Translations

You can search through objects in the repository to find existing translations for untranslated strings. This allows you to reuse existing translations for user interface objects that you have created or modified.

The LMU compares untranslated strings with string attributes of other objects in the repository. If it finds an object with the same string, it searches for a translation in the language that you have selected as the target language of the current LMU session. If a translation exists, the LMU displays the best candidate for translation and allows you to export it to a file.

For example, suppose you have selected English-American as your source language and Spanish as the target language. You have an applet with a title of Customer that has not been translated. After clicking the Find Translation button, the LMU searches through the repository for other objects with attributes of Customer. If it finds one, it looks for a Spanish translation of the string. If a translation already exists, the translation is displayed and you can export it to a file.

If the LMU finds more than one translation for a source string, the following rules apply:

- If the source string is an attribute of an object that is related to a business component, such as Control Caption or List Column Display Name, then translations from the same business component are examined first. If multiple translations exist in the same business component, the string that occurs the most is selected. If none of the translations exist in the same business component, then the translation that occurs the most often from among all business components is selected.

For example, suppose Applet A is based on the Account business component. Applet A contains a control caption with the value of *Account* and this value has been translated to *Account_FRA* for French. Now suppose you create a new applet, Applet B, that is also based on the Account business component and that also contains a control caption with the value of *Account*. When you run Find Translations, the LMU would find *Account_FRA* as an existing translation and select it as the best candidate for this string.

- If the source string is not an attribute related to a business component, such as Menu Item Caption, the translation that occurs the most is selected as the best candidate.

To find translated strings

- 1** Choose Tools > Utilities > Locale Management.

The Locale Management Utility appears.

- 2** Select the source language and the target language.
- 3** Select the applications or projects that you want to localize.
- 4** Select the Untranslated Strings tab.
- 5** Click the Find Translations button.

The LMU compares untranslated strings with strings of other objects in the repository. If other objects use the same source string, the LMU looks for existing translations of the string and displays the best candidates for translation in the Results window.

Exporting Strings and Other Locale-Specific Attributes

You use the Locale Management Utility to export strings and other locale-specific attributes to an external file.

To export strings and other locale-specific attributes

- 1 Choose Tools > Utilities > Locale Management.

The Locale Management Utility appears.

- 2 Select the Source and Target Languages.
- 3 Select the applications or projects that you want to localize.
- 4 Select the Export Tab.
- 5 Select whether you want to export Strings only or all localizable attributes.

All localizable attribute includes Strings and other layout attributes, such as the positions of controls. These attributes may be different for different locales.

- 6 Click Export.

When you click export the objects that you selected are exported to a file. The default directory for the file is the Tools/Objects directory of your Siebel installation. If you have chosen to export Strings Only, the strings are exported to a text file called “results.txt.” If you have selected to export All Localizable Attributes, Strings and other UI attributes are exported to a file called “results.slf.”

Importing Strings and Other Locale-Specific Attributes

You use the Locale Management Utility to import translated strings and other locale-specific attributes back into the repository. Use the preview functionality to see the results of the import process before you actually import them into the repository.

To preview the results of the import process

- 1 Choose Tools > Utilities > Locale Management.

The Locale Management Utility appears.

- 2 Select a source language and a target language.

- 3 Select the Import tab.
- 4 Enter the directory path and name of the file you are going to import.
- 5 Enter the path and name of the file where you want to store the results for previewing.

The default file name is “preview.txt.”

- 6 Click Preview.

The Locale Management Utility writes the results of the import process to the log file rather than to the repository.

NOTE: LMU will not mark changed records with a Redo flag when running in Preview mode.

To import strings and other locale-specific attributes into the repository

- 1 Choose Tools > Utilities > Locale Management.

The Locale Management Utility appears.

- 2 Select a source language and a target language.
- 3 Select the Import tab.
- 4 Enter the file name of the file from which you want to import locale-specific attributes.

You can also use the Browse button to find and select the file. The default file name is:

- `Results.txt` if the file contains strings only
- `Results.slf` if the file contains all locale-specific attributes

- 5 Select whether you want to mark records in the repository with the Redo flag that have changed since the export occurred.

When the import occurs, the LMU compares the source language records in the repository with the source language records in the import file. If the records in the repository have changed since the export occurred they are marked with the Redo flag. This helps you identify records that may need to be retranslated.

- 6 Click Import.

The locale-specific attributes are imported into the repository.

Identifying Modified Objects

You can use the Locale Management Utility to identify objects that have been modified in the repository since the last time you exported strings. This is useful when your development and localization efforts occur simultaneously. It helps you keep strings in the repository in sync with the strings that have been exported to a file for localization.

You can search for modified objects using the following two methods:

- Base your search on a specific date.
- Compare objects in the repository with objects in a source file, such as results.txt.

NOTE: When you base your search on a specific date, and run the search by clicking the Start button, all records returned for a modified project are marked as “Redo,” regardless of whether a particular locale attribute has changed. This is because the LMU searches for changes at the object level (the base record), not the attribute level.

To identify modified objects

- 1** Choose Tools > Utilities > Locale Management.

The Locale Management Utility appears.

- 2** Select a source language and a target language.
- 3** Select the Modified Objects tab.
- 4** Define the search criteria you want to use:
 - Select the Changed since check box and then specify a date after which you want to find modified objects.
 - Select the Different from file check box and then specify the file to compare the repository against.
- 5** Do one of the following:

- Click Start to find records that match the search criteria, display the results, and flag records returned in the search as Redo. Redo indicates that a record has been changed since the last time export occurred and therefore may need to be retranslated.
 - Click Preview to find records that match the search criteria and display the results. Preview does not mark records as Redo.
- 6** After you have identified modified objects, you can perform the following tasks:
- Click the Save button to save a result set in a log file.
 - Click the Go To button to open the Object Explorer and go to the parent object of the string or attribute.

NOTE: The Load button allows you to import a result set from a previously saved file. After loading the result set in the display window, you can perform Save or Goto operations on those records.

Using the LMU to Replace Strings

You can use the LMU to replace strings in a bulk mode. For example, suppose that you need change occurrences of Accounts to Companies for the English locale. You can use the LMU to export the strings to a file, manipulate the file so that it only contains the strings you want to replace, and then supply Companies as the string to replace Account. When using the LMU to replace strings, the source and target language are the same.

To use the LMU to replace strings

- 1** Identify the applications and/or projects to which the strings belong.
- 2** Export the strings you want to replace to an LMU file.

Use the procedure described in [“Exporting Strings and Other Locale-Specific Attributes” on page 605](#).

- 3** Remove strings from the LMU file that you do not want to replace.
- 4** In the Target String column of the LMU file, enter the string that you want to substitute for the original value.

- 5 Use the LMU to import the LMU file.
 - a Select source and target language (both are the same).
 - b Navigate to Import tab and specify the LMU file path.
 - c Click Import to replace the strings.

Running the LMU From the Command Line

You can run the LMU from the command line interface. Commands, syntax, usage and examples are proved in the following sections.

The syntax for the commands below use these conventions:

- `<xxx>` is a placeholder for a required parameter.
- `[xxx]` is a placeholder for an optional parameter.
- `xxx|yyy` is an selection parameter (that is, `xxx` or `yyy`)

NOTE: When specifying file names, the absolute path must be provided. For example, if you specify LMU file as `results.txt` for export, it will be created under the current directory. That is, assuming an installation directory of `d:\sea750\tools`, the file would created under `d:\sea750\tools\bin`, not `d:\sea750\tools\objects`.

Export Strings and Locale-Specific Attributes

Syntax `/lmu <srclang> <trglang> export <proj|app> <all|string> [<file>]`

Usage This command allows you to export localizable attributes for all projects or for all applications. If you specify *all*, then all attributes (translatable and language override attributes) are exported to a file with the extension of `.slf`; if you specify *string*, then string attributes only are exported to a file with a `.txt` extension. If file name is not specified, the exported file is named either `results.txt` or `results.slf`, and the file is exported to `/tools_root/objects/` directory.

Example `siebedev /u sadmin /p db2 /d server /lmu ENU FRA export proj all C:\temp\my_proj_results.txt`

This example instructs LMU to export all attributes (string and language override attributes) for all projects to a LMU file located at C:\temp, called my_proj_results.txt. The source language is English-American and the target language is French.

Import a LMU File

Syntax /lmu <srclang> <trglang> import <file>

Usage This command allows you to import a LMU file and mark all target locale objects as 'Redo' if the source string from the import file and the repository differ. You must specify the file name (with absolute path) to the import file.

Example siebdev /u sadmin /p db2 /d server /lmu ENU FRA import
D:\sea750\tools\objects\results.slf

This example instructs the LMU to import a file called results.slf from the folder D:\sea750\tools\objects. The source language of the LMU file is English-American (ENU) and the target language is French (FRA). The LMU file contains all localizable attributes (string and language override attributes).

Export Strings to be Translated

Syntax /lmu <srclang> <trglang> todo <proj|app> [<file>]

Usage This command allows you to export all untranslated strings and strings marked with 'Redo' flag to an LMU file. You can specify whether you want to export for all projects or all applications. The exported LMU file contains the related View Names.

Example siebdev /u sadmin /p db2 /d server /lmu ENU FRA todo app
D:\sea750\tools\objects\results.txt

This example instructs the LMU to find all untranslated strings and redo strings for all applications and export the results to D:\sea750\tools\objects\results.txt. The source language is English-American (ENU) and the target language is French (FRA).

Logical User Interface Objects Configuration

Using the Locale Management Utility

Pick Applets and Static Picklists 11

This chapter provides a conceptual overview of pick applets and picklists in Siebel applications. It also walks you through the steps in configuration process, using the Pick Applet and Pick List Wizards.

Pick Applets

Pick applets allow users to select a value from a list, and have the selection entered into controls or list column cells (Figure 166).

NOTE: In end-user documentation, pick applets are referred to dialog boxes.

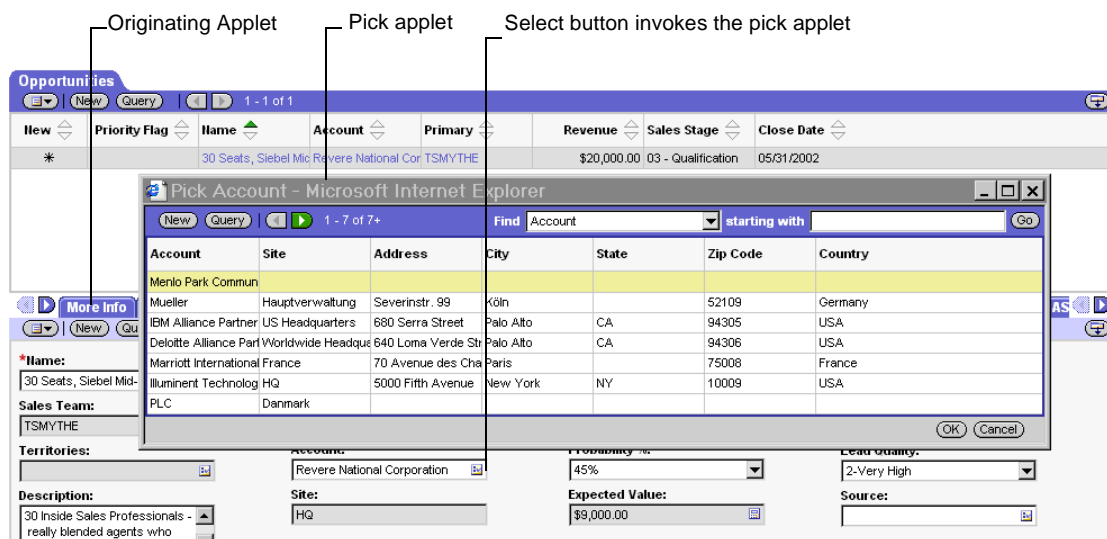


Figure 166. Pick Applet in a Siebel Application

Pick applets are invoked by clicking the Select button that appears next to certain fields. Pick applets contain a scrolling list table of available selections in one list column, with the information from related fields in adjacent list columns. The user selects a row in the list table and clicks the OK button to accept the selection. The pick applet is dismissed, and the user's selection populates the text box or list column cell in the originating applet (the applet from which the pick applet was invoked). The user's selection can also populate other controls or list column cells in the originating applet.

For example, when a user clicks the Select button in the Account field in the Opportunity Form applet, the Pick Account dialog box (pick applet) appears for the selection of an account (as illustrated in [Figure 166 on page 613](#)). Once an account has been selected, and the Pick Account dialog box has been dismissed, the Account text box contains the selected account, and the Site text box in the originating applet contains the site that corresponds to the selected account.

The data in the pick applet typically comes from a different business component than the data in the originating applet. There can be exceptions, for example, picking a parent Account for an Account or a parent Position for a Position record. When the user selects a record in the pick applet, the values in certain list columns in the selected record are copied to corresponding list columns in the originating applet. This is illustrated in Figure 167.

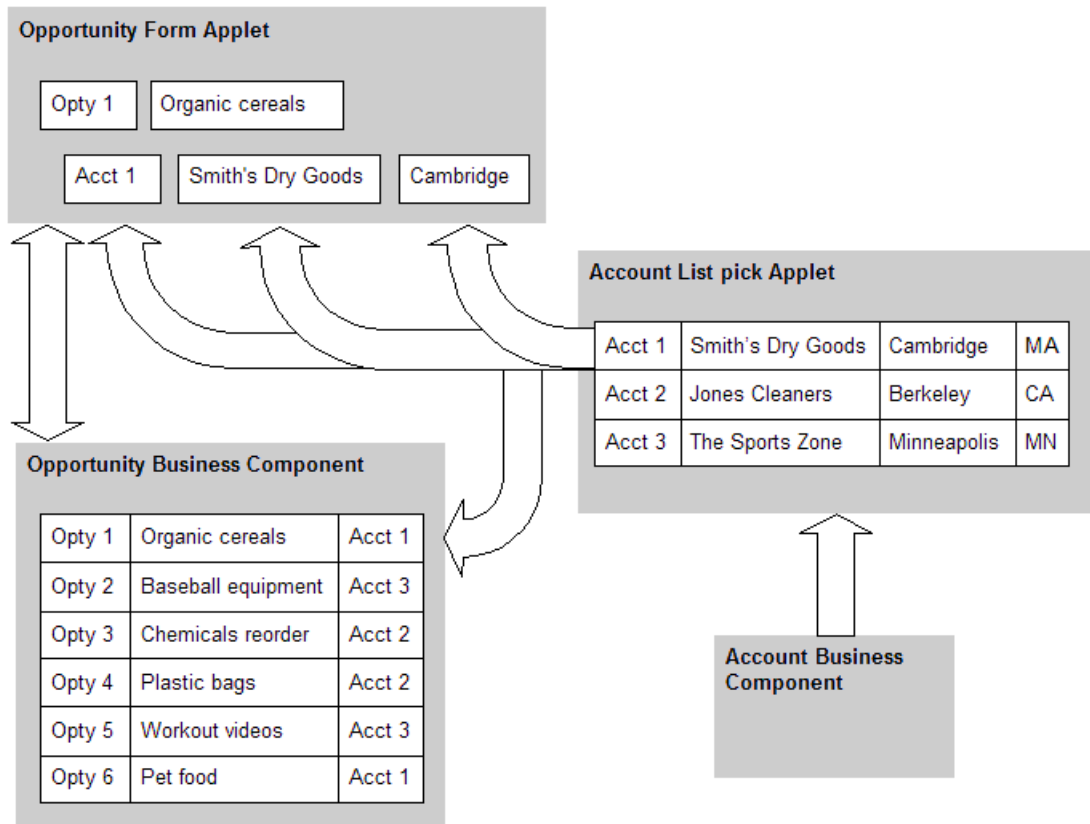


Figure 167. Data Flow in a Pick Applet

The following steps take place, from the user's perspective:

- 1** In the Opportunity Form applet, the user enters information for the Organic Cereals opportunity.
- 2** In the Opportunity Form applet, the user clicks the Select button.
- 3** The Account pick applet appears.
- 4** The pick applet displays rows from the Account business component.
- 5** The user selects Account 1, Smith's Dry Goods, in the pick applet, and then clicks OK.
- 6** Account data for Smith's Dry Goods populates controls in the Opportunity Form applet.

Pick applets maintain the foreign keys that facilitate join relationships. In the opportunity and account example, there is a foreign key in the Opportunity business component identifying the account for each opportunity. When the user selects an account in the pick applet, it populates this foreign key field. This selection associates the account with this opportunity for future use by the join that uses the foreign key.

Pick applets are implemented using object types illustrated in [Figure 168](#).

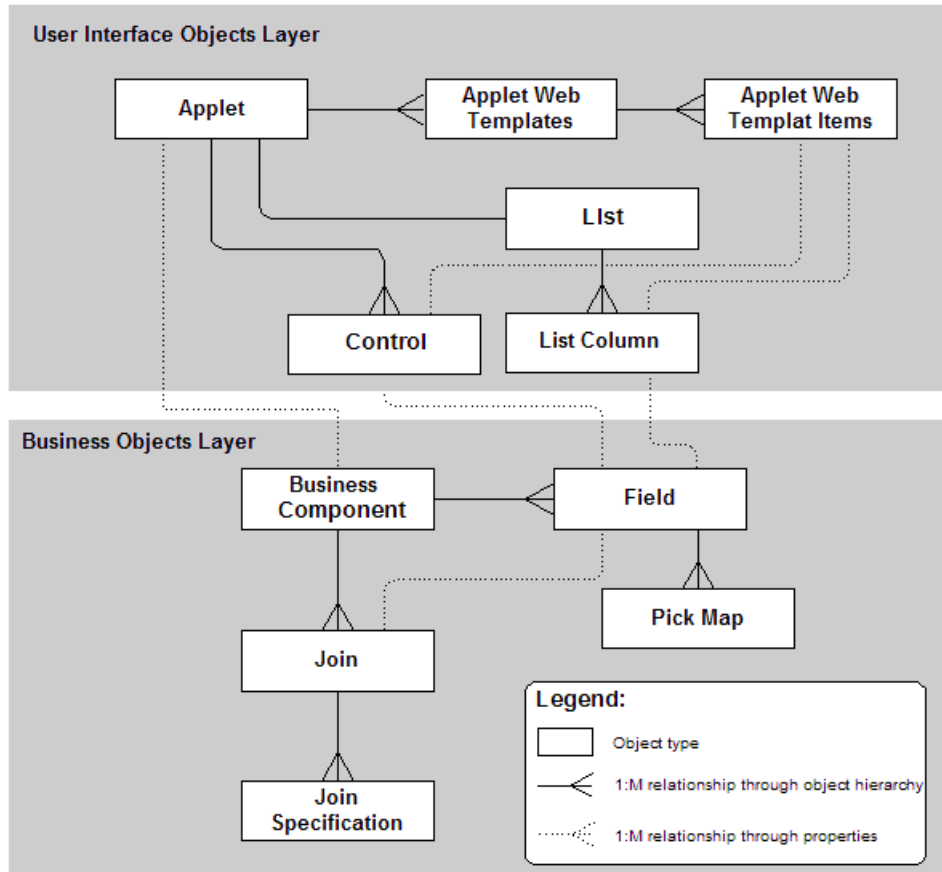


Figure 168. Pick Applet Architecture

[Figure 168](#) shows the object definitions used in the implementation of a pick applet in greater detail, and identifies the interrelationships.

The roles of the object definitions in [Figure 168 on page 617](#) are summarized in the following list and discussed in greater detail in the subsequent subsections. The pick applet example referenced is the Account pick applet illustrated in [Figure 166 on page 613](#).

- **Originating applet.** Contains the control or list column that invokes the pick applet. After the pick applet is invoked and a value is selected, specific controls in the originating applet display revised values. In the example the originating applet is the Opportunity Form Applet.
- **Pick applet.** Dialog box that is invoked for the selection of a value. The dialog box is a list applet containing scrolling list table rows. Each row corresponds to a business component record. In the example, the pick applet is called Account pick applet.
- **Originating business component.** Business component of the originating applet. This business component (in the example, the Opportunity business component) supplies the data presented in the originating applet (Opportunity form applet). The selection process in the pick applet results in the update of the current record in this business component.
- **Pick business component.** Business component of the pick applet. Data from fields in this business component is displayed in the list columns of the pick applet. In the example the pick business component is Account.
- **Originating control or originating list column.** Appears in the originating applet. When you click the originating control or list column, it invokes the pick applet. In the example, the originating control is the Account control.
- **Originating field.** Field in the originating business component that the originating control represents. It has pick map child object definitions that define the mapping of fields from the pick business component into the originating business component. In the example the originating field is the Account field.
- **Pick List.** Referenced by the field of the originating control, and identifies the pick applet's business component. In the example, the pick list object is called PickList Opportunity Account.

- **Pick maps.** Children of the originating field. Each pick map object definition defines a correspondence between a field in the pick business component and one in the originating business component. These correspondences provide the information required to update the current originating business component record with information from the pick business component record as soon as a record is picked.

When a user selects a value from an unbounded picklist, it is copied to the field with which the picklist is associated using the corresponding pick map that references the same field. Fields associated with other child pick maps are only populated when the picklist is bounded.

NOTE: Typing a new value into an unbounded picklist does *not* automatically add it to the list of values that can be picked.

Fields in pick map objects are NOT updated when a user picks a value from an unbounded picklist. Any applet based on CSSBuscomp or CSSBCBase with an unbound picklist will not map all the values in the pick map. For all the values in a pick map to be mapped, the picklist must be bounded.

- **Join and join specification.** Child object definition of the originating business component. The join specification is a child of the join and is referenced by the join field. One of the pick maps updates the join field. A change in the value of the join field results in the update of all fields whose values are derived from the join. This update is not as immediate as the update performed by the pick maps. In the absence of the other pick maps, the data would not be updated until the user left the view and returned to it. In the example the join is called S_ORG_EXT and the join specification is Account Id.

Configuring the Originating Applet

The originating applet contains the control or list column that invokes the pick applet. It may also contain other controls or list columns that are populated by the user's selection from the list applet. The originating applet itself requires no special configuration.

Figure 169 is a detail of the originating applet in Figure 168 on page 617.

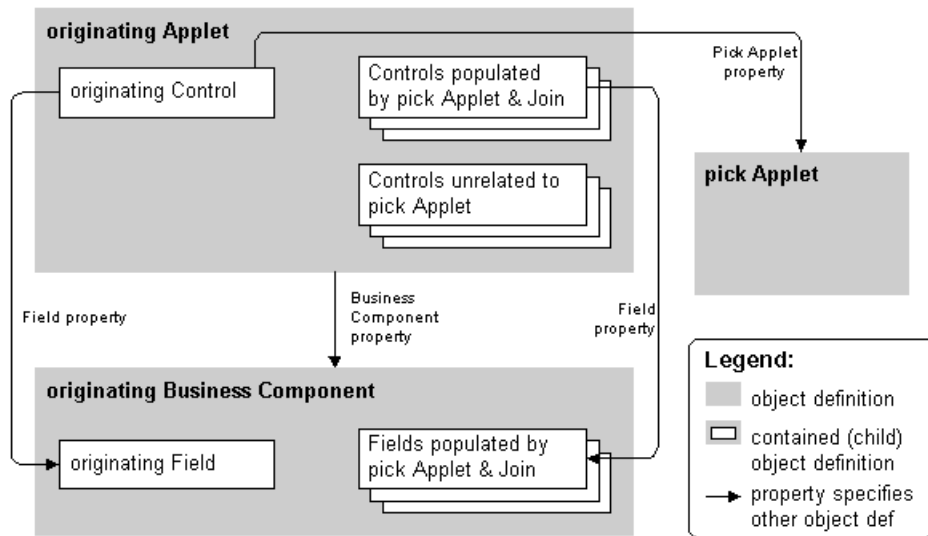


Figure 169. Originating Applet Details

As indicated in Figure 169, the important property setting for the originating applet is as follows:

- **Business Component.** Creates the association between the originating applet and the originating business component.

The important child object definitions of the originating applet are the following:

- **Originating control.** Invokes the pick applet, as the result of the user's clicking the drop-down icon. The originating control has the name of the pick applet in its Pick Applet property. The field specified in the Field property of the originating control is the originating field, and has pick map child object definitions, as discussed in [“Configuring Originating Business Components” on page 643](#).

The control or list column must have its Runtime property set to TRUE.

- **Controls populated by the pick applet.** Each control for which some field in the originating business component is populated by a pick map object definition will be updated when the user makes a selection from the pick applet.
- **Controls unrelated to the pick applet.** Other controls in the applet.

Configuring the Pick Applet

The pick applet is the dialog box that is invoked for the selection of a value. It is a list applet containing a scrolling list of rows. Each row corresponds to a business component record in the pick business component.

Figure 170 shows a detailed definition of the pick applet from Figure 168 on page 617.

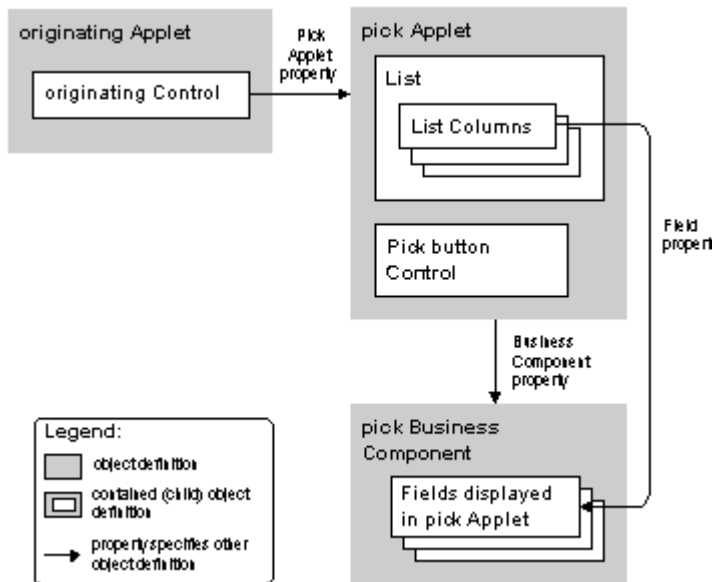


Figure 170. Pick Applet Details

The pick applet (Applet object type) has the following important property settings:

- **Business Component.** Pick business component.
- **Class.** CSSFrameList, indicating that this is a list applet.
- **Type.** A value of Pick List is entered, to indicate that this is a pick applet. This setting configures the behavior of the dialog box and button controls.

- **Title.** Name of the pick applet that appears in the title bar.

The pick applet has the following important child object definitions:

- **List.** List columns are attached to the list.
- **List columns (grandchild object definitions).** Each displays the contents of one field in the business component.
- **Pick Record control.** Invokes the PickRecord method when clicked. The PickRecord method locates the pick map child object definitions of the originating field and, from these, determines which fields to update in the originating business component. These fields are updated based on the record selected from the pick business component by the user.
- **Web Templates.** Define the layout, such as position of list columns and controls, for each of the defined modes.
- **Web Template Items.** Map list columns and controls to placeholders in the Web template. Web template items exist for each list column and control defined for the applet.

Using the Pick Applet Wizard

You can use the Pick Applet Wizard to create a pick applet.

To configure applets using the Pick Applet Wizard

- 1 From Siebel Tools main menu, choose File > New Object.

The New Object Wizards dialog box appears.

- 2 Click the Applets tab.

- 3 Click the Pick Applet icon, and then click OK.

The General dialog box appears.

- 4 In the General dialog box, enter information for the following and then click Next.

- Project

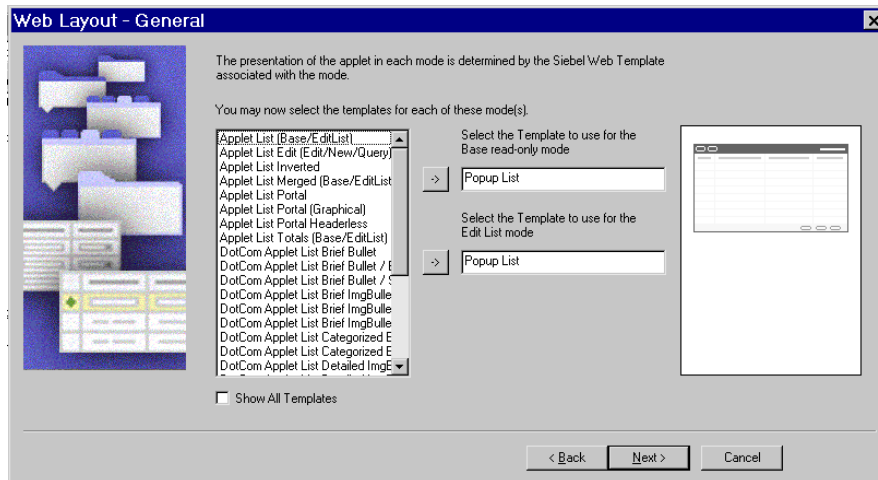
Pick Applets and Static Picklists

Using the Pick Applet Wizard

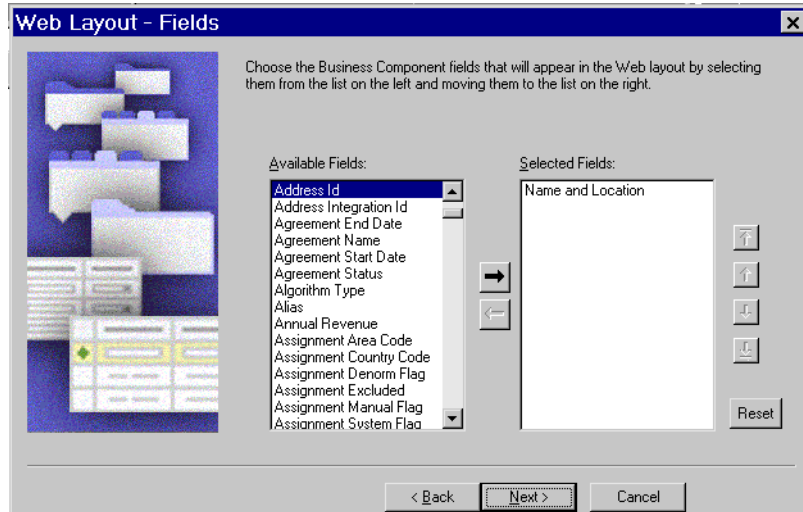
- Pick business component
- Name for the Picklist Applet
- Display Name

The Web Layout General dialog box appears.

- 5 In the Web Layout General dialog box, select the templates to use for the Base and Edit List modes, and then click Next.



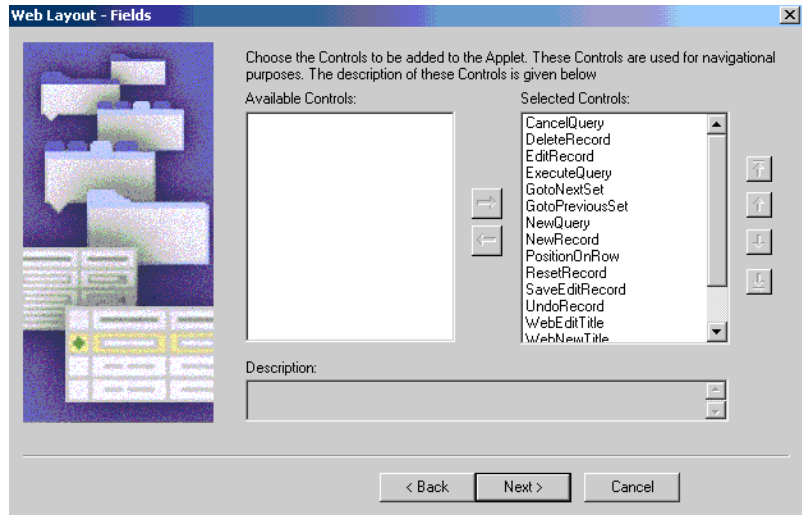
- 6 In the Web Layout - Fields dialog box, select the fields you want to appear in the pick applet, and then click Next.



Pick Applets and Static Picklists

Using the Pick Applet Wizard

- 7 In the Second Web Layout - Fields dialog box, select the controls you want to appear in the pick applet and then click Next.



NOTE: By default all the controls are present in the Selected Controls box. If you wish to deselect any of these controls, highlight them and click the left-facing arrow to move these controls into the Available Controls box. The controls that appear by default are based on the Model Pick Applet in the Siebel repository.

- 8 In the Finish dialog box, review the information, and then click Finish.

The Pick Applet Wizard creates the pick applet object, and then opens the Web Layout editor that you can use to map list columns and controls to the placeholders in the Web Template.

See [“Editing the Web Layout of Views” on page 566](#) for more information.

Configuring the Originating Business Component

The originating business component is the business component of the originating applet, as specified in the Business Component property of the Applet object. This business component supplies the data presented in the originating applet. The selection process in the pick applet results in the update of the current record in this business component. [Figure 171 on page 628](#) shows the detailed definition of the originating business component from [Figure 168 on page 617](#).

Pick Applets and Static Picklists

Using the Pick Applet Wizard

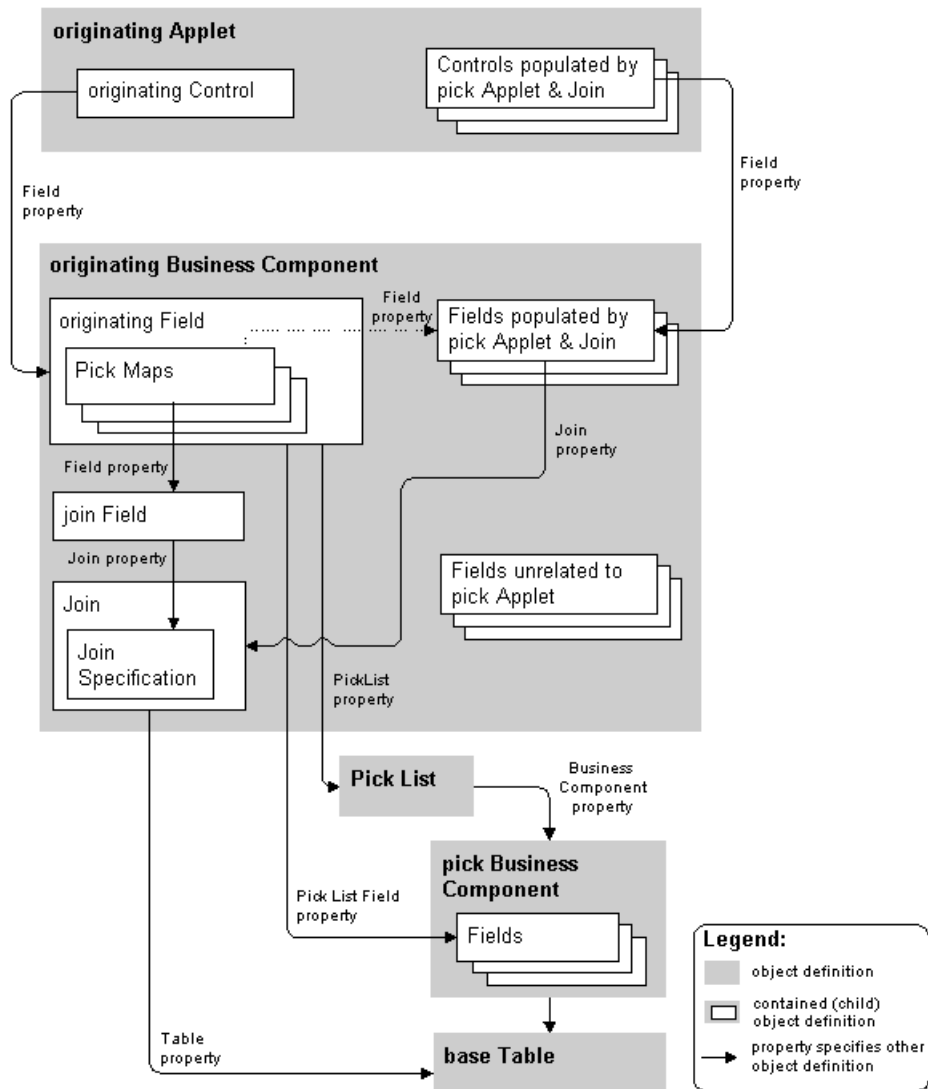


Figure 171. Originating Business Component Details

The originating business component has no important property settings that are related to its role in the pick process.

The originating business component has the following important child object definitions:

- **Originating field.** The originating control displays the data from this field. The originating field has no special role other than being the parent of the pick map object definitions. The Pick List property of the Field specifies the Pick List object. In the Siebel application architecture, pick maps are children of an originating field, rather than the originating business component, in order to support pick applets on more than one field in the business component.

NOTE: The originating field should be a field based on a database column. Pick applets and picklists cannot be associated with read-only fields, including calculated fields.

- **Pick maps.** Children of the originating field. Each pick map defines a correspondence between a field in the pick business component and one in the originating business component. These correspondences provide the information required to immediately update the current originating business component record with information from the pick business component record when a record is picked. Additionally, one of the pick maps updates the join field, and eventually this causes the join to update the fields in the business component that are dependent on the join.

NOTE: Test your pick map definition after creating it. If the originating field stays the same after choosing a value from the pick applet, you should check the pick map definition for that field.

Each Pick Map object definition has two important properties:

- **Field.** Identifies a field in the (grandparent) originating business component that is to be populated by data from a field in the pick business component, when the PickRecord method is invoked.

- **Pick List Field.** Identifies a field in the pick business component that is the source of data for the field in the Field property of the Pick Map object.

Fields in Pick Map objects are updated when the user picks a value from an unbounded picklist. However, fields in Pick Map objects are not updated by the picklist when the user types in a new value (the field the user typed something into is, however, obviously updated with the user's entry).

Typing a new value into an unbounded picklist does *not* automatically add it to the list of values that can be picked.

Do not define more than one multi-value field in the originating business component that maps to the same destination field in the pick applet's underlying business component (Pick List Field Property). Doing so causes the drop-down arrow for the picklist not to show; as a result, users will not be able to use the picklist.

- **Join field.** Serves as a foreign key in the join used by the pick applet. Typically, the join field contains Id in its name, such as Account Id or Key Contact Id. It is identified in the Source Field property of the join specification. The join field is one of the fields identified in a pick map object definition. When the user selects a record from the pick applet, the join field is updated (because of the pick map in which it is identified), and this results in the update of all fields that are based on the join.

NOTE: Fields in the originating business component, and the controls or list columns that represent them, initially are updated by the action of the pick maps. The join and join specification do not update the contents of the applet until the user leaves the view and returns to it.

- **Join and join specification.** The join and join specification object definitions set up the join between the base tables of the originating and pick business components. This join populates those fields in the originating business component that have this join's name in their Join property.

- **Fields populated by the pick applet and join.** Fields that have the join's name in their Join property are updated when the join field's value changes. Fields that are identified in the Field property of Pick Map object definitions are updated when a selection is made from the pick applet. There is some overlap in the roles of the pick maps and join, in that both generally update the same fields, but the action of the pick maps is immediate and that of the join is somewhat delayed.

That is, even though pick maps can update the display value of joined fields (for example, Account Name) when the user picks a record, pick maps do not physically copy a value to the joined fields—only to the foreign key field (for example, Account Id).

Constraining a Picklist

You can dynamically filter a pick applet to display only records that have field values matching corresponding fields in the originating business component's records. This is called constraining a picklist. For example, a Contact's pick applet invoked from an applet that displays quotes could be configured to display only contacts for the current quote's account.

Pick applet constraints are defined using the Constrain property in the Pick Map object type. For example, if you want to configure a Country picklist to display only states that are part of that country, you need a way to indicate the relationship between each state and its country. You could use the existing Description field in the Picklist Generic BusComp to do this or alternatively, you could extend the table and use a new column. Next, you would need to fill the Description field with valid Country values. Use one of the following methods to accomplish this.

- From the client application, choose Site Map > Application Administration > List Of Values View and populate the Description field with valid Country values.
- From Siebel Tools, choose Account BusComp > State Field > PickMap. Insert a new record in the PickMap list and set the following properties:
 - Field = Country
 - Constrain = True
 - Pick List Field = Description

After a user selects a value from the Country picklist, the State picklist appears. The values in the State picklist are constrained by the value selected from the Country picklist. The value chosen from the Country picklist is used to filter the values that appear in the State picklist. Only the values where the Description field contains the selected value from the Country picklist will appear in the State picklist.

Pick maps can be of two types: copy pick map or constraint pick maps. Copy pick maps perform the role described in [“Configuring Originating Business Components” on page 643](#): a copy pick map updates the current originating business component record with information from the pick business component. A constraint pick map also configures a mapping between the originating and pick business components, but its purpose is different. It is used to filter the list of records displayed in the pick applet to present only those that have matching values in corresponding fields in the originating and the pick business component.

A pick map is configured as a constraint pick map by setting its Constrain property to TRUE. If FALSE (which is the default), the pick map is a copy picklist.

The pick applet displays only contacts with the same Account, Account Id, and Account Location as the quote. To accomplish this, define a constraint pick map as a child object of the Contact Last Name field (in addition to the various copy pick map object definitions provided in order to implement pick behavior). The presence of this constraint pick map indicates to the system that it is to filter the displayed records in the pick applet.

NOTE: If the constrained field refers to a joined table in the pick business component, the foreign key field must also be constrained. Otherwise, a “This operation is not available for read-only field” error will occur if a new record is created in the pick applet.

Configuring Pick Business Components

The pick business component is the business component of the pick applet. Data from fields in this business component is displayed in the list columns of the pick applet.

Figure 172 shows a detailed definition of the pick business component from Figure 168 on page 617.

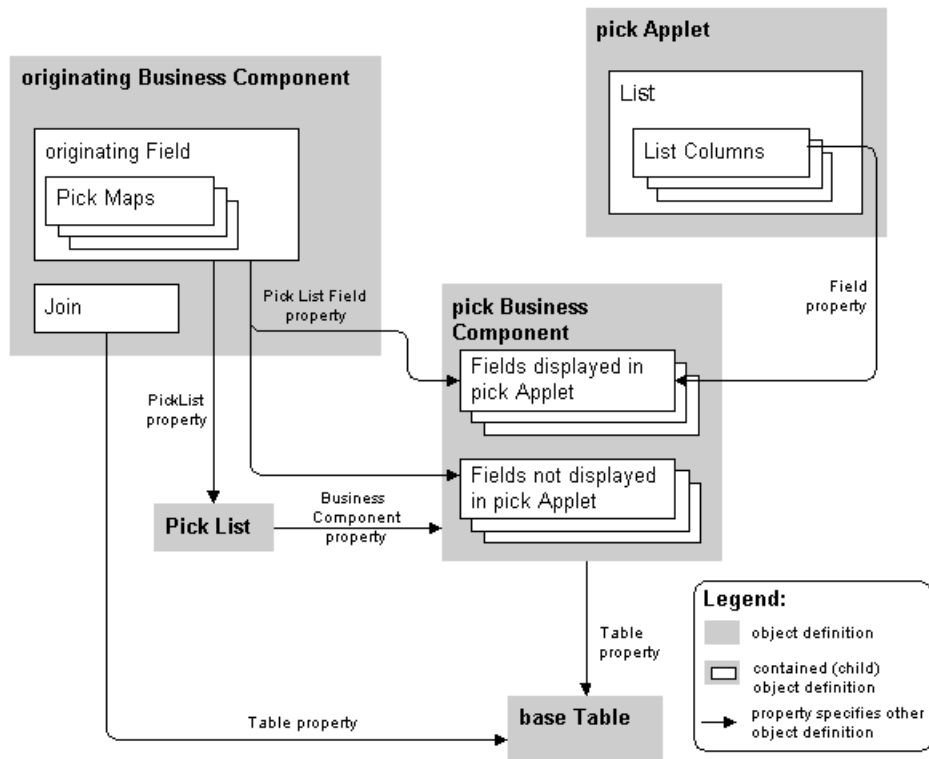


Figure 172. Pick Business Component Details

The pick business component has no important property settings with respect to its role in the pick process.

The pick business component has the following important child object definitions:

- **Fields displayed in the pick applet.** Populate the list columns in the pick applet. They are referenced in the Field property of corresponding list columns in the pick applet. Some of the same fields may be identified in the Pick List Field property of Pick Map object definitions and, hence, have a role in updating corresponding fields in the originating business component.
- **Fields not displayed in the pick applet.** Although not displayed in list columns in the pick applet, some of these fields may be identified in the Pick List Field property of Pick Map object definitions and therefore have a role in updating corresponding fields in the originating business component.

Configuring Picklists

The field of the originating control references the Pick List object definition. The Pick List object definition identifies the pick business component. In this way, the identity of the pick business component is made known to the pick applet.

Figure 173 shows the detailed definition of the Pick List object definition from Figure 168 on page 617.

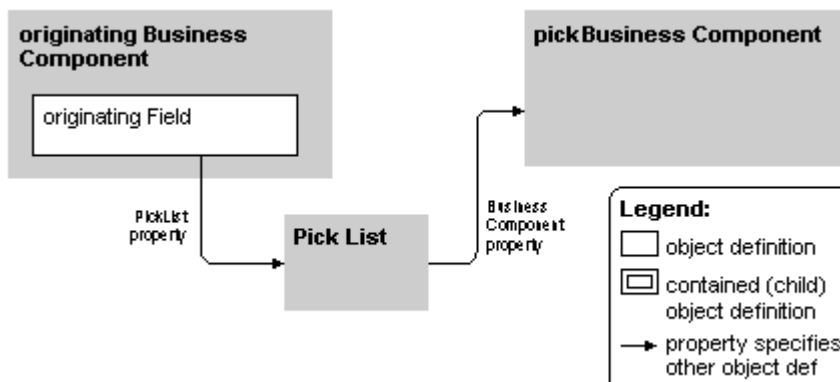


Figure 173. Pick List Details

The Pick List object definition has the following important property, when used in pick applet configuration: Business Component. This property identifies the pick business component.

NOTE: When configuring a pick applet invoked from a multi-value group applet, define the picklist on the originating field in the originating business component, not on fields in the multi-value group business component. For more information on multi-value group applets, refer to [Chapter 12, “Multi-Value Group and Association Applets.”](#)

Creating a Picklist Using the Pick List Wizard

You can use the Pick List Wizard to create dynamic picklists.

To open the Pick List wizard

- 1 From the Tools main menu, choose File > New Object.

The New Object Wizards dialog box appears.

NOTE: You can also invoke the Pick List wizard by selecting the field for which you would like to create a picklist, right-clicking, and choosing Add Pick List.

- 2 Select the Pick List icon, and then click OK.

The Pick List Wizard appears.

- 3 In the Pick List dialog box, enter the following information and then click Next.

- Project
- Business Component (originating business component; the parent business component of the field that will display the picklist)
- Field

- 4 In the Pick List Type dialog box, select Dynamic.

NOTE: Static picklists draw their values from a predefined list of values (LOV). Dynamic picklists draw their values from a business component. For more information about static pick lists, see [“Creating a Static Picklist Using the Pick List Wizard” on page 645](#).

- 5 In the Pick List Definition dialog box, choose whether you want to create a new picklist or use an existing one:

- If you want to create a new picklist, select the Create a New Pick List radio button, and then click Next.

The Pick List Definition dialog box appears. Go to [Step 6](#).

- If you want to use an existing picklist, select the Use Existing Pick List radio button, select the picklist from the Pick List box and then click Next.

The Pick Map dialog box appears. Skip to [Step 7](#).

- 6** In the Pick List Definition dialog box, enter the information for the picklist, and then click Next.
 - Business Component (pick business component)
 - Sort field in the picklist
 - Name
 - Search Specification (not required)
 - Comment (not required)

- 7** In the Pick Map dialog box, select the source field in the originating business component and the target field in the pick business component, and then click Add.

The selected fields are displayed in the Current Map window.

- 8** Click Next, verify the information in the Finish Dialog box, and then click Finish.

Static Picklists

A static picklist is a selection list that is invoked from a particular text box or list column in an applet. A static picklist in a Siebel application is shown in [Figure 174](#).

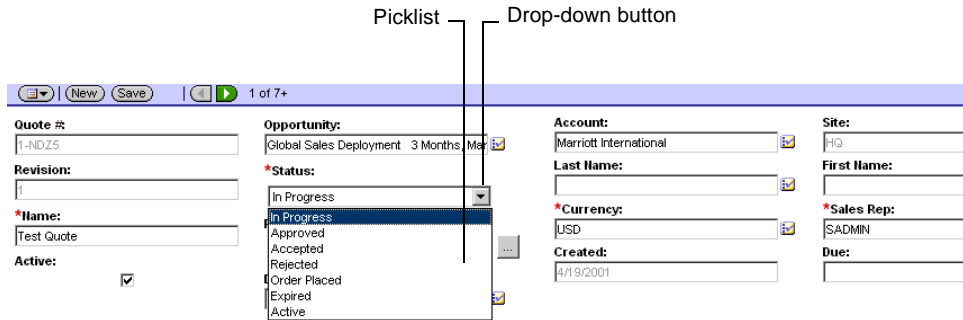


Figure 174. Static Picklist in a Siebel Application

When the user clicks the drop-down button to the right of the text box, a single-column picklist appears. The user selects a value from the list by clicking the desired value. The selected value replaces the previous value in the text box.

NOTE: You cannot delete the lookup value. You can set the picked field (for example, Lead Quality) back to NULL, unless it is required.

Here is how a static picklist compares with a dynamic pick applet:

- They are similar in that a static picklist allows the user to select a value to populate a text box with data.
- They are different in that a static picklist does not draw values dynamically from a pick business component. A static picklist is a static list of available selection values. Configuration of these values is an administration activity that is performed in the List of Values Administration view in a Siebel application.

- They are different in that a static picklist generally does not invoke a dialog box with multiple list columns and buttons. All that appears is a simple one-column pop-up list, without buttons.

NOTE: It is possible to use a pick applet rather than a simple drop-down list to display a static list of values, but this is not common practice.

- They are different in that a static picklist does not populate multiple controls in the originating applet. It populates a single control in the applet, and the corresponding field in the underlying business component.

Static picklists are implemented using object types illustrated in [Figure 175](#).

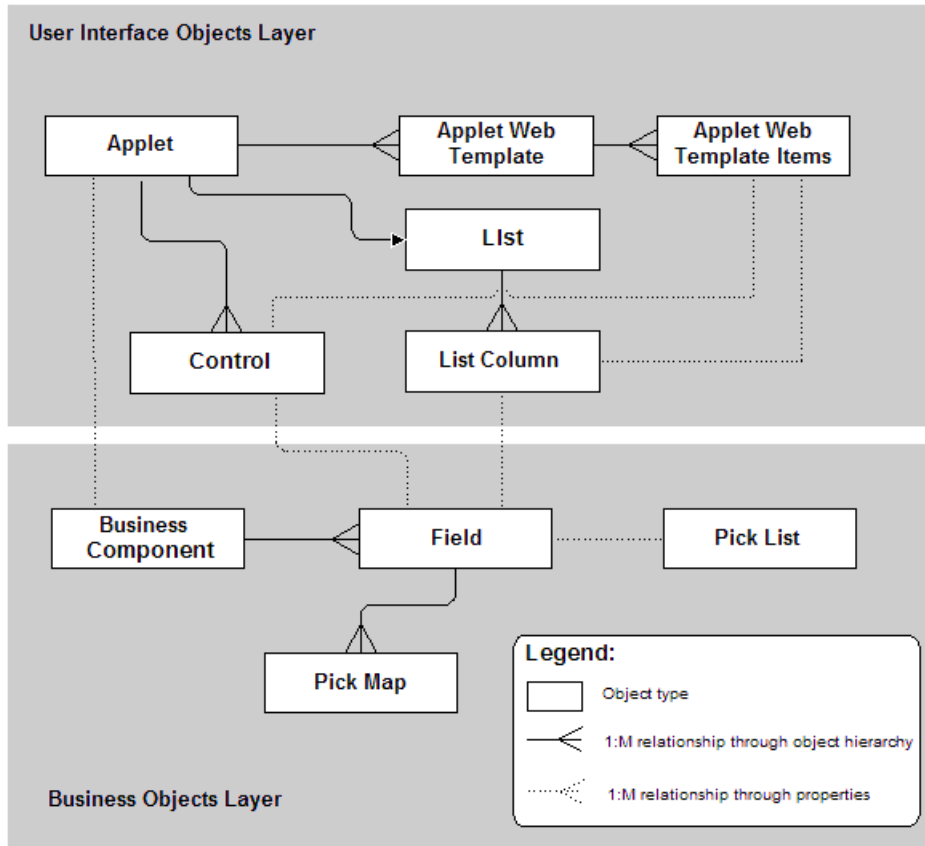


Figure 175. Static Picklist Architecture

Figure 176 shows the object types used in the implementation of a static picklist in greater detail, and identifies their interrelationships.

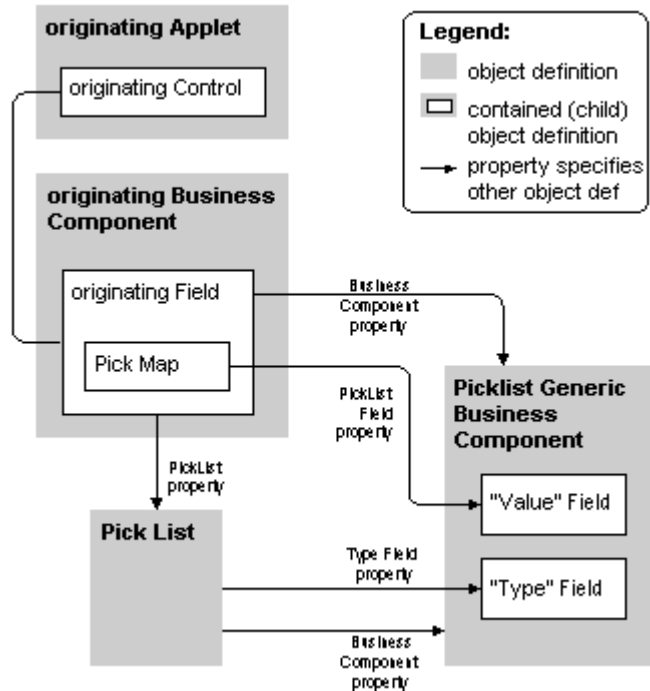


Figure 176. Static Picklist Details

The roles of the object definitions in Figure 176 are summarized in the following list, and discussed in greater detail in the subsequent sections. The static picklist example refers to the Quality picklist illustrated in Figure 174 on page 638.

- **Originating applet.** Contains the control or list column that invokes the picklist. After the selection of a value from the picklist, the originating control displays a revised value. In the example, the originating applet is the Opportunity form applet.

- **Originating business component.** Business component of the originating applet. This business component (in the example, the Opportunity business component) supplies the data presented in the originating applet (Opportunity form applet). The selection process in the picklist results in the update of one field in the current record in this business component.
- **Originating control or originating list column.** Appears in the originating applet. It initiates the picklist when clicked. In the example, this is the Quality control.
- **Originating field.** Field in the originating business component that the originating control represents. Generally, it has one pick map child object definition that defines the mapping of a field from the PickList Generic business component into the originating business component. In the example, the originating field is Quality.

NOTE: If the originating field is a custom field, make sure that it can accommodate the LOV table values. A field shorter than the LOV table values will cause truncation when it is displayed or stored in the database.

- **PickList Generic business component.** Special-purpose business component for the list-of-value lists that are used in static picklists. It is administered through the List of Values view in the System Administration screen in Siebel applications. To access the List of Values view choose Site Map > Application Administration > List of Values from the menu bar.
- **Pick List object.** The field of the originating control references the Pick List object definition. The Pick List object definition identifies the pick applet's business component, which is always PickList Generic. In the example, the Pick List is called Picklist Quality.
- **Pick Map object.** Child of the originating field. The pick map defines a correspondence between the Value field in the PickList Generic business component and the originating field. This correspondence provides the information required to update the current originating business component record with information from the PickList Generic business component record when a selection is made.

- **Sequence Property.** Defines the sequence for updating fields in the current originating business component record with information from pick business component record when picking this record. If you do not define sequence numbers on pick maps, they will be executed in the order in which they were created.

Configuring Originating Applets

The originating applet (Applet object type) has the following important properties:

- **Business Component.** Identifies the originating business component.

The originating control (Control object type) or list column (List Column object type) has the following important properties:

- **Field.** Identifies the originating field in the originating business component.
- **Pick Applet.** Leave blank for a static picklist.
- **Runtime.** Set to TRUE to indicate that a static picklist is attached, and needs to be activated in response to a user click on the control or list column.

Configuring Originating Business Components

The originating business component is the business component of the originating applet. The data value selected from the pick applet updates the value in the originating field of this business component.

The originating business component has no essential properties for the configuration of a static picklist. However, the field (child) and pick map (grandchild) object definitions are significant.

The originating field is specified in the Field property of the originating Control or List Column object. It has the following important properties:

- **PickList.** Identifies the Pick List object definition.

The originating field has one important child object definition, the Pick Map object. Unlike dynamic picklists, static picklists generally have exactly one Pick Map object definition. The Pick Map object has the following important properties:

- **Field.** Contains the name of the originating (parent) field.

- **Pick List Field.** In this property enter “Value.” This setting references the Value field in the PickList Generic business component.

NOTE: You would use multiple pick maps only if you use a multiple column selection list.

Configuring the Pick List

The Pick List object is referenced by the originating field and identifies the business component and field that populate the pick applet. The Pick List object definition has the following important properties:

- **Business Component.** In this property enter the value PickList Generic. This indicates that the list of values comes from the system tables.
- **Type Field.** In this property enter the value Type. This indicates that Type is the field in the PickList Generic business component to search for types. Each list of values has a type, which uniquely identifies the list and each value in it.
- **Type Value.** In this property enter the relevant type for the list of values. For example, in the Lead Quality picklist in [Figure 174 on page 638](#), the values that appear in the list have a Type field value of LEAD_QUALITY in the List of Values View in Siebel applications.
- **Search Specification.** If a Search Specification value is defined for the Pick List, it overrides the business component’s Search Specification. If a Search Specification is not defined, the Search Specification for the business component is used. The default value of the Search Specification is blank.
- **Sort Specification.** If a Sort Specification value appears in the Pick List object definition, this overrides the business component’s sorting with that of the Pick List. The default value for the Sort Specification property is blank, which tells the system to use the business component’s sorting.

This feature is useful for non-standard sorting of values in a static picklist that is based on a list of values in the PickList Generic business component. By default, a list of values is sorted in ascending order on the Order By field within a Type. If the Order By values are blank, the entries for the Type are alphabetically sorted on the Value field, in ascending order. You can alter this behavior for one static picklist by setting a sort specification in its picklist.

- **No Insert.** Static picklists must have their No Insert property set to TRUE to work properly. If this property is set to FALSE the application generates the following error message:

`"Unable to create picklist popup applet."`

Creating a Static Picklist Using the Pick List Wizard

You can create a static picklist by using the Pick List Wizard.

To create a static pick list using the Pick List Wizard

- 1 From the Tools main menu, choose File > New Object.

The New Object Wizards dialog box appears.

NOTE: You can also invoke the Pick List wizard by selecting the field for which you would like to create a picklist, right-clicking, and choosing Add Pick List.

- 2 Select the Pick List icon, and click OK.
- 3 In the Pick List dialog box, enter the following information, and then click Next.
 - Project
 - Business Component (originating business component; the parent business component of the field that will display the picklist)
 - Field
- 4 In the Pick List Type dialog box, select Static, and then click Next.

NOTE: Static picklists draw their values from a predefined list of values (LOV). Dynamic picklists draw their values from a business component. For more information about dynamic picklists, see [“Creating a Picklist Using the Pick List Wizard” on page 636](#).

- 5 In the Pick List Definition dialog box, do one of the following:

Pick Applets and Static Picklists

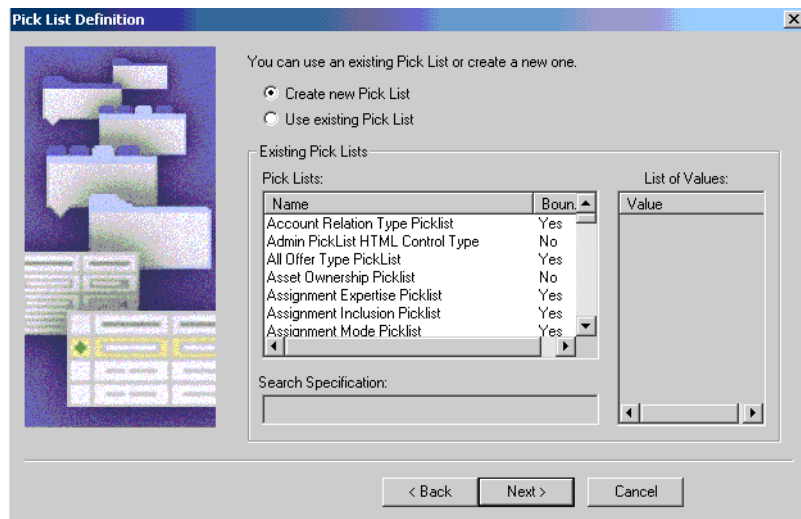
Static Picklists

- If you want to create a new picklist, select the Create New Pick List radio button and then click Next.

The Pick List Definition dialog box appears.

- If you want to use an existing picklist, select the picklist and associated list of values you want to use, and then click Next.

The Finish dialog box appears. Go to [Step 8](#).



- 6 If you are creating a New List of Values, do the following:
 - a Enter a unique name for the picklist.
 - b Select the Create New List of Values radio button, and then click Next.
 - c In the List of Values dialog box, enter a name for the List of Values and the Values.

For more information about List of Values, see *Applications Administration Guide*.
 - d Click Next.

- 7** If you are using an existing List of Values, do the following:
 - a** Enter a unique name for the picklist.
 - b** Select the Use predetermined List of Values radio button.
 - c** Select the List of Values Type, and then click Next.
 - d** In the third Pick List Definition dialog box, enter a search specification, a comment, and select whether you want the picklist to be bounded.
 - e** Click Next.
- 8** In the Finish dialog box, review the specifications for the picklist, and then click Finish.

The PickList Generic Business Component

The PickList Generic business component is a specialized business component reserved for lists of values for static picklists. The data in the Picklist Generic business component looks something like [Table 41](#).

Table 41. Example of Data in Picklist Generic Business Component

Type Field Contents	Value Field Contents
LEAD_QUALITY	Excellent
LEAD_QUALITY	Very Good
LEAD_QUALITY	High
LEAD_QUALITY	Fair
LEAD_QUALITY	Poor
PERSON_TITLE	Mr.
PERSON_TITLE	Ms.
PERSON_TITLE	Dr.

Table 41. Example of Data in Picklist Generic Business Component

Type Field Contents	Value Field Contents
ACCOUNT_TYPE	Commercial
ACCOUNT_TYPE	Competitor
ACCOUNT_TYPE	Customer

Two of the fields in the Picklist Generic business component together define and group the lists of values, as follows:

- **Type.** Each list of values has a type. The type groups together all records that are in one list of values. For example, a type of LEAD_QUALITY identifies a record as a member of the Lead Quality list of values, and the type ACCOUNT_TYPE refers to the Account Type list of values.
- **Value.** The Value is the portion of the record that actually appears in the static picklist. For example, Lead Quality values are Excellent, Very Good, High, Fair, and Poor.

Hierarchical Picklists

A hierarchical picklist displays values that are constrained by values selected in another picklist. For example, in the Service Request Detail Applet, the Area and Subarea fields are both picklists that draw their values from the List of values table (S_LST_OF_VAL). The items available in the Subarea picklist depend on what the user has selected in the Area picklist.

The hierarchical relationship between the values is established in the list of values table. All the values for picklists within the hierarchy are defined using the same LOV Type. For example, for Area and Subarea, the values are defined using the SR_AREA LOV Type.

The Parent Independent Code column is used to specify a parent value. For example, consider the following example LOV shown in [Table 42](#).

Table 42. Sample LOVs for Hierarchical Picklist

Type	Display Value	Language Independent Code	Parent LIC
SAMPLE_LOV	1	1	
SAMPLE_LOV	A	A	1
SAMPLE_LOV	B	B	1
SAMPLE_LOV	2	2	
SAMPLE_LOV	C	C	2
SAMPLE_LOV	D	D	2

Assume two picklists are configured to display the values shown in [Table 42](#) in a hierarchical relationship. One picklist is the parent picklist, and the other picklist is the child. The parent picklist displays the values {1, 2}. If the user selects 1, the values displayed in the child picklist are {A, B}; If the user selects 2, the values displayed in the child picklist are {C, D}.

Implementing a hierarchical list of values also involves configuration work. You must configure two picklists to support this hierarchical relationship. The parent picklist must be based on the PickList Hierarchical business component, and the child picklist must be based on the PickList Hierarchical Sub-Area business component.

To implement a hierarchical picklist

- 1** Configure a parent and a child picklist.
 - a** Set the Business Component property of the parent picklist to Picklist Hierarchical.
 - b** Set the Business Component property of the child picklist to Sub-Area Picklist Hierarchical.
- 2** Go to the business component that contains the fields that you want to associate the hierarchical picklists with.

- a** Set the Picklist property of the parent field to the parent picklist.
- b** Set the Immediate Post Changes property of the parent field to TRUE.
- c** Set the PickList property of the child field to the child picklist.
- d** For the child field, create the following Pick Map objects.

Field	PickList Field	Constrain
[name of parent field]	Parent	TRUE
[name of child field name]	Value	

- 3** Compile changes to a repository file.
- 4** Add LOV values using the Parent LIC column to designate the parent value.

See table above for a simple example and see the Constrained Lists of Values section in *Applications Administration Guide* for detailed discussion.

- 5** Test.

Pop-Up Visibility Rules

You can limit picklist visibility by organization or responsibility using the following properties of the Business Component object type:

- **Popup Visibility Auto All.** Valid values are TRUE, FALSE, and blank.
- **Popup Visibility Type.** Valid values are:
 - All
 - Personal
 - Sales Rep
 - Manager
 - Organization
 - Sub-Organization

- Group
- Catalog

If Popup Visibility Auto All is set to TRUE and the user has access to any “All” views, pop-up visibility will be All. If the user does not have access to any “All” views, pop-up visibility will be the value of Popup Visibility Type.

If Popup Visibility Auto All is set to TRUE, the user does not have access to any “All” views, and Popup Visibility Type is blank, pop-up visibility will default to the most restrictive value defined for the business component, for example Sales Rep.

If Popup Visibility Auto All is blank, pop-up visibility will be the value for Popup Visibility Type.

If both properties are blank, pop-up visibility will default to the most restrictive value defined for the business component.

For more information on visibility rules, see *Security Guide for Siebel eBusiness Applications*.

Working With Lists of Values

After configuring a new picklist, you may need to add a new list of values to display in the picklist. For example, if you are customizing an applet and want to provide a picklist field that contains a list of values that is not already available in your Siebel application, you can create a new list of values.

Every static picklist includes a type property as part of its definition in Siebel Tools. For example the picklist that allows a user to select the personal title for a contact or an employee has a Type Value property of MR_MS, which in turn is associated with a list of values that includes Miss, Ms, Mrs., Mr., and Dr. To create a new list of values for a custom picklist, you first create a record that defines the new list-of-values type, in which the type corresponds to the type property of a picklist object. You then add records to define each value that will be available for that type.

NOTE: If you want to see the additions or changes to a list of values in real time, you need to clear the cache.

You can create a new lists of values using Siebel Tools or using the List of Values view in the Siebel Web client. For information about working with existing lists of values, see *Applications Administration Guide*.

To create a new list of values using Siebel Tools

- 1 Choose Screens > System Administration > Lists of Values.

Lists of values are displayed in the Object List Editor.

- 2 Enter new records for the List of Values.

Some of the fields are described in the table below. For a complete description of LOV fields, see *Applications Administration Guide*.

Field	Description
Type	The name of the list of values type, for example REAL_ESTATE_TYPE. This value is used to group all other values for this type. The value defined in this field must match the value defined in the Type Value property of the picklist that is configured to display these values.
Display Value	Value displayed in the picklist.
Language Independent Code	Typically the same value as the American English version of the display value. See “About the Language Independent Code” on page 655 .
Language Name	Name of the language for the Display Value.

For information about fields used for multilingual lists of values, such as Translate, Multilingual, and Language-Independent Code, see [“Multilingual Lists of Values” on page 654](#) and [“Administering the Multilingual List of Values” on page 684](#).

To create a new list-of-values type using the List of Values view in the Web client

- 1** From the application-level menu, choose View > Site Map > Application Administration > List of Values.
- 2** Click New to create a new record, and then enter LOV_TYPE in the Type field.

This type is used specifically when you want to define a new list of values.

Enter the text specified as the type property in Siebel Tools. For example, to create a new list of values that will contain real estate types, the type property might be REAL_ESTATE_TYPE. The text you enter here will become the text used for this list-of-values type.

- 3** In both the Display Value field and the Language Independent Code field, enter the name of the list-of-values type.

The Display Value is the value displayed to the user in the picklist.

The Language-Independent Code is typically the American English version of the display value. For more information about the role of Language Independent Code in a multi-lingual list of values, see [“Multilingual Lists of Values” on page 654](#) and [“Administering the Multilingual List of Values” on page 684](#).

- 4** Enter a value in the Language Name field.

After you enter this value, the record has all required information. This list of values has now been created as a new list-of-values type.

Add the values you want this list to contain by continuing with the next steps.

- 5** Click the New button to create another record and enter REAL_ESTATE_TYPE in the Type field.

This will be the first value for the list-of-values type REAL_ESTATE_TYPE.

- 6** In the Display Value field and the Language Independent Code field, enter a value that you want to display in the REAL_ESTATE_TYPE list.

For example, you might enter Apartment.

- 7** Enter a value in the Language Name field.

- 8** If applicable, in the Order field, you can enter a number to indicate the numerical order in which the value appears in a drop-down list.

For example, if you plan to create REAL_ESTATE_TYPE records for Apartment, House, Condo, and Commercial, you might want the order numbers to be 1 for Apartment, 2 for House, 3 for Condo, and 4 for Commercial.

- 9** If appropriate, fill in the remaining fields in the record.

For a complete description of these fields, see *Applications Administration Guide*.

- 10** To add another value to the list, repeat Step 5 through Step 9.

Multilingual Lists of Values

You can configure your Siebel application to display multilingual lists of values (MLOV) in static picklists. This allows you to display values in the active language of the user. It also allows the values selected by a user in one language to be retrieved by users working in other languages.

See *Global Deployment Guide* for more information about the user's active language.

NOTE: To enable MLOV, the static picklist must be bounded and must not be a hierarchical picklist.

An example of a multilingual picklist appears in [Figure 177](#). When the user clicks the drop-down arrow, a list appears with values displayed in the user's active language that the user can select to populate the field.

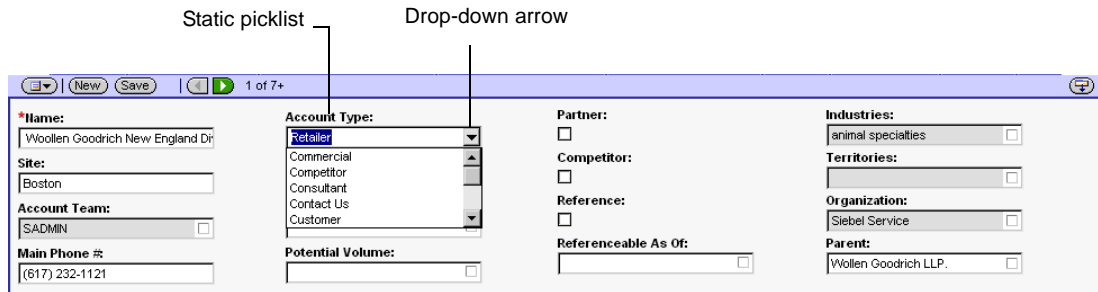


Figure 177. Static Picklist

About the Language Independent Code

The LOV table contains a Display Value column and a Language Independent Code column. Both monolingual and multilingual lists of values display values from Display Value column to the user. However, after the user selects a value in a picklist, the actual value stored in the database is different for monolingual and multilingual lists of values.

- A monolingual picklist stores the Display Value.
- A multilingual picklist stores the Language Independent Code.

For example consider the values in [Table 43](#). A multilingual picklist would display the Display Value (Mr., Señor, or Herr) depending on the active language of the user, but it would store the value Mr. in the database, because that is the value defined in the Language Independent Code column.

Table 43. Example LIC

Display Value	Language Independent Code
Mr.	Mr.
Señor	Mr.
Herr	Mr.

NOTE: Generally the language-independent code value is the same as the American-English version of a particular selection value.

Storing the value from LIC column rather than the Display Value column allows the data to be stored in a form that users working in other languages are able to retrieve and allows the roll up of data for management reports, regardless of the language of the users who enter the data.

CAUTION: The length of the language-independent code (the value stored in the database) must be equal to the longest display value for the MLOV. If it is not, the display value will be truncated. If the standard column does not meet your requirements and you are using a custom extension column, the column must be VARCHAR and have a maximum length (width) of 30.

Enabling the Multilingual List of Values

To set up an MLOV operation, you must modify the LOV configuration in Siebel Tools, as well as perform administration tasks. For MLOV to be enabled the following conditions must be met:

- The column of the field using the picklist has the Translation Table property set to S_LST_OF_VAL
- The picklist must be bounded (Lov Bounded Property is set to True)
- The picklist must use the same LOV Type as specified in the Lov Type property of the column.

NOTE: A picklist's LOV Type should always match the LOV Type of the underlying column (the column on which the picklist's field is based).

You enable MLOVs by completing the following tasks.

- 1 [“Identifying Which Columns to Enable” on page 658.](#)
 - a [“Making Sure the LOV Type Is Translatable” on page 659.](#)
 - b [“Determining If the Picklist Is Bounded” on page 660.](#)
- 2 [“Configuring the Multilingual List of Values in Siebel Tools” on page 663.](#)
- 3 [“Adding Translated Display Values in Application Administration” on page 664.](#)
- 4 [“Upgrading Existing Data Using the MLOV Upgrade Utility” on page 665.](#)
- 5 [“Recompiling and Deploying” on page 673.](#)

These steps are discussed in the following sections.

NOTE: Configuration of MLOVs can impact performance, especially when the field on which the picklist is based is used as part of a search or sort. Performance characteristics should be considered and verified in conjunction with configuration of MLOVs.

Identifying Which Columns to Enable

Not every list of values type can be enabled as multilingual. You need to determine which columns you can enable based on the LOV type. LOV types must meet the following conditions:

- The column must be marked as translatable. See [“Making Sure the LOV Type Is Translatable” on page 659](#).
- The picklist must be bounded. See [“Determining If the Picklist Is Bounded” on page 660](#).
- The column must not be one of the [“Special Cases.”](#) See [“Special Cases” on page 662](#).

NOTE: Do not set up a column for a MLOV unless you are sure that you intend to use that column for your implementation.

Columns storing data that is read by server programs, such as Assignment Manager, Siebel Remote, Siebel Anywhere, or Workflow Manager, require additional configuration. See the following for more information:

- [“Configuring Siebel Business Process Designer to Use MLOV-Enabled Fields” on page 676](#)
- [“Configuring Siebel Assignment Manager to Use MLOV-Enabled Fields” on page 681](#)
- [“Configuring Siebel Anywhere for Use with MLOV-Enabled Fields” on page 683](#)

Configuring MLOVs may also include changes to the Siebel Visibility Rules. Any reference in a Visibility Rule to an LOV entry for a type you plan to configure for multilingual support must be changed from the Display Value to the language-independent Code. Check the visibility rules for references to any LOV entries as part of your configuration of MLOVs.

NOTE: Custom extension columns can always be MLOV enabled.

To check visibility rules

- 1** In Siebel Tools, navigate to the Dock Object Visibility Rules view.
Use the flat screen view to simplify searching.
- 2** Go to the SQL Statement field and search for literals across all rows that are not null.
- 3** Examine the results for values that need to be translated.

NOTE: Changing visibility rules requires the assistance of Siebel Expert Services.

Making Sure the LOV Type Is Translatable

A *translatable* type is a list of values type that can be modified, or translated into additional languages, without affecting the functionality of your application. This is indicated in the Translate setting in the List of Values Administration view in Siebel eBusiness Applications.

If an item is translatable, it can be modified without affecting Siebel eBusiness Applications functionality.

To determine if an LOV type is translatable

- 1** Connect to the server database using Siebel eBusiness Applications.
- 2** From the application-level menu, choose View > Site Map > Application Administration > List of Values.
- 3** For the LOV type that you are interested in, look in the Translate list column for a check mark.

If you add a LOV type, set this list column according to your configuration. Do not change any existing settings shipped with Siebel eBusiness Applications, because these are set to reflect the Siebel eBusiness Applications configuration. Changing this setting will not allow you to enable an LOV type.

Determining If the Picklist Is Bounded

A *bounded picklist* is a picklist where users must choose from the existing choices and cannot enter their own data. An *unbounded picklist* is a picklist where users may either enter their own data or choose from the existing choices. Only bounded picklists can be configured to be multilingual.

You must verify that the picklist is bounded and that the underlying column has the Lov Bounded property set to True.

To determine if a picklist is bounded from the Picklists list

- 1 Connect to the server database using Siebel Tools.
- 2 Select the Flat tab in the Object Explorer.
- 3 Select the Pick List object type.
- 4 Query the Type Value property for the list of values type you are interested in.

If the Bounded property is checked, then that item is a bounded picklist.

To determine if a picklist is bounded from the Columns list

- 1 Connect to the server database using Siebel Tools.
- 2 Select the Flat tab in the Object Explorer.
- 3 Select the Column object type.
- 4 Query the LOV Type property for the list of values type you are interested in.

If the LOV Bounded property has a check mark, then the picklist for that target column is bounded.

NOTE: The *Translate* property of the column is only for internal use, and has nothing to do with the configuration of MLOVs.

All columns for a particular LOV type must be bounded. If any of the columns for the LOV type is not bounded, then none of the columns can be set to multilingual for that LOV type.

For example, [Table 44](#) shows the columns for the LOV type AVAILABILITY_STATUS. Although three of the columns are LOV bounded, you would not be able to enable these columns as multilingual, because one column (NEXT_AVAIL_CD) is Unbounded. If you were to run the MLOV Upgrade Utility, you would receive an error message that says the columns are inconsistently bounded. See [“About the MLOV Upgrade Log File” on page 671](#) for more information.

Table 44. Example of Inconsistently Bounded Columns

Name	LOV Type	LOV Bounded
CURR_AVAIL_CD	AVAILABILITY_STATUS	Y
NEXT_AVAIL_CD	AVAILABILITY_STATUS	Y
CURR_AVAIL_CD	AVAILABILITY_STATUS	Y
NEXT_AVAIL_CD	AVAILABILITY_STATUS	N

The Lov Bounded and Lov Type properties in the column object are read-only for standard columns in Siebel eBusiness Applications, but are editable for columns that have been added using the Database Extension Designer. If you add columns, you must set both the Lov Bounded and the Lov Type properties for each column individually, consistent with your configuration.

NOTE: You need to modify the Lov bounded property for standard Siebel columns, you must contact Siebel Expert Services to determine the feasibility of this change.

Special Cases

There are special cases that should be considered when determining whether an LOV TYPE can be enabled for multilingual display. The columns listed in [Table 45](#) cannot be MLOV enabled.

Table 45. Columns That Cannot Be MLOV Enabled

Table	Column
S_AGREE_POSTN	APPR_ROLE_CD
S_CONTACT	PREF_LANG_I
S_CONTACT_X	ATTRIB_48
S_CS_RUN	STATUS_CD
S_DOC_ORDER	TAX_EXEMPT_REASON
S_ONL_LAYOUT	CONTROL_TYPE_CD
S_ORG_EXT	DIVN_CD
S_ORG_EXT	DIVN_TYPE_CD
S_ORG_EXT_XM	NAME
S_PRI_LST_ITEM	PRI_METH_CD
S_PROD_INT_CRSE	CRSE_TYPE_CD
S_PROD_INT_X	ATTRIB_50
S_PROD_INT_X	ATTRIB_51
S_PROD_INT_X	ATTRIB_53
S_PROJ_ORG	PROJ_ROLE_CD
S_PROJITEM	PROD_AREA_CD
S_PROJITEM	STATUS_CD
S_SRC	SRC_CD
S_SRC	STATUS_CD

Table 45. Columns That Cannot Be MLOV Enabled

Table	Column
S_SRC_EVT	FORMAT_CD
S_SRCH_PROP	NAME

Configuring the Multilingual List of Values in Siebel Tools

Once you have determined which columns to enable as multilingual, you configure those columns in Siebel Tools.

List of values types are enabled for multilingual support on a target-column basis. Because a list of values type can be used for different target columns, the Multilingual property must be implemented for all target columns that use the same type.

The following procedure describes the process for manually enabling a column. The list of values type ACCOUNT_TYPE is used as an example.

To enable a column for multilingual storage and display

- 1 Open Siebel Tools, and connect to the server database.
- 2 Select the Flat tab in the Object Explorer.
- 3 Select the Column object type.
- 4 Choose Query > New Query, enter the name of the desired list of values type in the LOV Type property, and then press ENTER to execute the query.

In this example, you would search for an LOV Type of ACCOUNT_TYPE. The query shows you the columns that use that LOV type, in this case there is only one column named OU_TYPE_CD.

Name	Parent Table	LOV Type	LOV Bounded	W	Changed	Project	User Name
OU_TYPE_CD	S_ORG_EXT	ACCOUNT_TYPE	✓			Newtable	Account Type

- 5 Check that all the columns using that type have a check mark in the LOV Bounded property.
- 6 Change the Translation Table Name to S_LST_OF_VAL for all the columns returned by the query.

NOTE: Changing the properties of object definitions directly on the server is a nonstandard practice that is used only for configuration of columns for multilingual storage and display. Under all other circumstances, the correct and safe way to change object definition properties is to check out projects to the local repository, make the desired changes, and check them back in to the server. See [Chapter 18, “Application Development Projects,”](#) for more information about checking in and checking out projects.

Adding Translated Display Values in Application Administration

Once you have configured a column to be multilingual, display values must be defined for each language that will be supported.

To add translated display values

- 1 Using the Siebel Web client, connect to the server database.
- 2 From the application-level menu, choose Site Map > Application Administration > List of Values.
- 3 Find the list of values type for the enabled target columns.
- 4 For every language that will be supported, create a new record for each display value for that list of values type.

For instance, if you plan to support German and French in addition to the existing English display values, create two new records for each display value—one in German and one in French.

- 5 For each new record, the language-independent code must be the same as for the original record, but the entries in the Language and Display Value list columns are set differently, as appropriate.

For more information on adding and maintaining translated values, refer to [“Administering the Multilingual List of Values” on page 684.](#)

Upgrading Existing Data Using the MLOV Upgrade Utility

After you have configured your application for use with MLOVs and added new display values for all the languages you intend to support, you must upgrade your existing LOV data. You do this using the MLOV Upgrade Utility.

NOTE: Even if you have just completed a new installation of your Siebel application, you must perform this data upgrade.

You run the MLOV Upgrade Utility in two modes.

- **Validation.** Running the utility in this mode validates the current repository for data inconsistencies. If the utility finds inconsistencies, the program stops and writes the errors to a log file.
- **Translation.** Running the utility in this mode:
 - Changes data in target columns that are configured for MLOVs from the display value to the language-independent code.
 - When you set the target column for an LOV Type to multilingual, the utility sets the MULTI-LINGUAL flag in the LOV table (S_LST_OF_VAL) to make sure of consistency between the multilingual state of the target column and its corresponding List of Values in the LOV table (S_LST_OF_VAL).
 - Verifies that all target columns using the desired MLOV type have been enabled.

NOTE: Target columns are columns that store either the display value or the language-independent code as part of user data.

The MLOV Upgrade Utility upgrades target columns that are marked as bounded and updates list of values types that are not already marked as multilingual. You can run the utility as often as you need to; only data that has not already been upgraded will be affected.

NOTE: The upgrade process run by MLOV Upgrade Utility is not reversible.

Running the MLOV Upgrade Utility

You run the MLOV Upgrade Utility using the Siebel Software Configuration Utility. The Siebel Software Configuration Utility is a wizard that will help you define the required parameters for running the MLOV Upgrade Utility. You run the utility in validation mode first, fix errors as they appear, and then run it in translate mode, which will enable your existing data for MLOVs.

NOTE: Before running the MLOV upgrade, drop all indexes from the columns that you are upgrading. Once the MLOV upgrade is complete, recreate the indexes.

To run the MLOV Upgrade Utility in a Windows Environment

- 1 Start the Siebel Software Configuration Utility by choosing Start > Siebel Enterprise Server > Configure DB Server.

NOTE: You can also start the Siebel Software Configuration Utility from the DOS Prompt command line. See [“To start the MLOV Upgrade Utility from the DOS Prompt” on page 668](#).

The Siebel Software Configuration Utility appears.

- 2 Enter the required parameters to run the MLOV Upgrade Utility in validation Mode.

See [Table 46](#) for a list of the wizard dialog boxes, options, and required values.

When you run the MLOV Upgrade Utility, it checks for errors and writes them to a log file. The default name of the log file is `mlovupgd_verify.log` and the default location is the `siebsrvr\LOG` directory.

- 3 Review the log file and resolve errors as necessary.
See [“About the MLOV Upgrade Log File” on page 671](#) for more information.
- 4 If an error is detected, resume running MLOV Upgrade Utility in validation mode by using the DOS Prompt to navigate to the BIN directory of your Siebel Server root directory (`SIEBEL_ROOT\BIN`), and then at the command prompt typing:

```
siebugg /m master_mlov_verify.ucf
```

The MLOV Upgrade Utility resumes running.

- 5 Repeat [Step 1](#) through [Step 4](#) until no errors are detected.
- 6 Start the Siebel Software Utility following the steps described in [Step 1](#).
- 7 Enter the required parameters to run the MLOV Upgrade Utility in translation mode.

See [Table 46 on page 670](#) for a list of the wizard dialog boxes, options, and required values.

The MLOV Upgrade Utility enables your existing data for MLOV. For columns configured for MLOVs, the MLOV Upgrade Utility finds LOV values in user data that are not in S_LST_OF_VAL and inserts them into S_LST_OF_VAL as inactive. It changes the display value of bounded columns to the language independent code and sets the value for the Multilingual attribute to true.

To run the MLOV Upgrade Utility in a UNIX Environment

- 1 Start the Siebel Software Configuration Utility doing the following:
 - Navigate to the Siebel root directory and type `source siebenv.csh`
 - Type `setenv LANGUAGE DISPLAY_LANGUAGE` (where `DISPLAY_LANGUAGE` represents the three letter code for the display language; for example `ENU`, `FRA`, `DEU`, and so on).
 - Type `setenv SIEBEL_ROOT SIEBEL_ROOT` (where `SIEBEL_ROOT` is the name of the directory where you installed the Siebel Server).
 - Type the command `dbsrvr_config.ksh`

The Siebel Software Configuration Utility appears.

- 2 Enter the required parameters to run the MLOV Upgrade Utility in validation Mode.

See [Table 46](#) for a list of the wizard dialog boxes, options, and required values.

When you run the MLOV Upgrade Utility, it checks for errors and writes them to a log file. The default name of the log file is `mlovupgd_verify.log` and the default location is the `siebsrvr\LOG` directory.

- 3 Review the log file and resolve errors as necessary.

See [“About the MLOV Upgrade Log File” on page 671](#) for more information.

- 4 If an error is detected, resume running MLOV Upgrade Utility in validation mode by navigating to the bin directory of your Siebel Server root directory (`SIEBEL_ROOT/bin`), and then typing the following command:

```
svrupgwiz /m master_mlov_verify.ucf
```

The MLOV Upgrade Utility resumes running.

- 5 Repeat [Step 1](#) through [Step 4](#) until no errors are detected.
- 6 Start the Siebel Software Utility following the steps described in [Step 1](#).
- 7 Enter the required parameters to run the MLOV Upgrade Utility in translation mode.

See [Table 46 on page 670](#) for a list of the wizard dialog boxes, options, and required values.

The MLOV Upgrade Utility enables your existing data for MLOV. For columns configured for MLOVs, the MLOV Upgrade Utility finds LOV values in user data that are not in `S_LST_OF_VAL` and inserts them into `S_LST_OF_VAL` as inactive. It changes the display value of bounded columns to the language independent code and sets the value for the Multilingual attribute to true.

To start the MLOV Upgrade Utility from the DOS Prompt

- 1 From the DOS Prompt, navigate to the `\BIN` directory of your Siebel Server root directory.

For example: `cd siebsrv/BIN`

- 2 Run the MLOV Upgrade Utility in validation mode by typing the following at the command prompt:

```
ssincfgw -l language_code -v Y
```

where *language_code* is the three-letter code (all capitals) for the language in which you want to display the GUI.

For example, to run the MLOV Upgrade Utility in English, you would type:

```
ssincfgw -l ENU -v Y
```

The Open dialog box appears.

- 3 Select `dbsrvr.scm` and then click Open.

The Siebel Software Configuration Utility -DB Server Configuration dialog box appears.

Resume Running MLOV Upgrade Utility

In case of an error, you can resume running the MLOV Upgrade Utility in validation mode or translation mode.

To resume the MLOV Upgrade Utility in a Windows environment

- 1 At the DOS Prompt, navigate to the BIN directory of your Siebel Server root directory (`SIEBEL_ROOT\BIN`)
- 2 At the command prompt do one of the following:
 - To resume running in validation mode, type `siebug /m master_mlov_verify.ucf`
 - To resume running in translation mode, type `siebug /m master_mlov_translate.ucf`

To resume the MLOV Upgrade Utility in a UNIX Environment

- 1 Navigate to the bin directory of your Siebel Server root directory (`SIEBEL_ROOT/bin`).
- 2 At the prompt, do one of the following:

- To resume running in validation mode, type `srvrupgwiz /m master_mlov_verify.ucf`
- To resume running in translate mode, type `srvrupgwiz /m master_mlov_translate.ucf`

MLOV Upgrade Utility Parameters

To run the MLOV Upgrade Utility, complete the dialog boxes listed in [Table 46](#) and enter or select the values as you go.

Table 46. MLOV Upgrade Utility

In This Dialog Box	Enter Or Select The Following
Siebel Enterprise Parameters: Gateway Server Address	Gateway Server Address Enterprise Server Address
Installation and Configuration Parameters: Siebel Server Directory	Siebel Server Directory
Installation and Configuration Parameters: Siebel Database Server Directory	Database Server Directory
Database Server Options: Siebel Database Operation	Run Database Utilities
Database Utilities: Database Utility Selection	Multi-lingual List of Values Conversion
MLOV Parameters: MLOV Operation	Validate or Translate, depending on the mode you want to run.
Installation and Configuration Parameters: Language Selection	Base language of your Siebel application.
Installation and Configuration Parameters: RDBMS Platform	RDBMS Platform
Installation and Configuration Parameters: ODBC Data Source Name	ODBC Data Source Name
Installation and Configuration Parameters: Database User Name	Database User Name Database Password
Installation and Configuration Parameters: Table Owner	Table Owner Name Table Owner Password

Table 46. MLOV Upgrade Utility

In This Dialog Box	Enter Or Select The Following
MLOV Parameters: Repository Name	Repository Name
Configuration Parameter Review	Review the parameters you have defined and then click Finish

About the MLOV Upgrade Log File

After the utility runs in either validation mode or translation mode, it writes any errors to a log file. The default names of the log files are `mlovupgd_verify.log` and `mlovupgd_translate.log`. The files are located in the `siebsrvr/LOG` directory.

- **LOVs Inconsistently Bounded or Translation Table Property Not Set to S_LST_VAL.** The message that appears in the log file for LOVs that have the bounded property on columns where they are used set inconsistently (one bounded and one not bounded) or LOV domains that do not have the Translation Table property set to `S_LST_VAL` is the following:

The following Validation checks for:

1- Two or more columns defined in the same LOV domain are inconsistently bounded (one bounded, one not)

2- Two or more columns are defined in the same LOV domain and at least one of them does not have a Translation Table Name of `S_LST_OF_VAL`.

Any errors of these types are listed in the log file. The information listed includes the LOV Type, Column, and Table.

To fix the LOV types that appear in the log file

- 1 Open Siebel Tools, and connect to the server database.
- 2 Select the Flat tab in the Object Explorer.
- 3 Select the Column object type.
- 4 Choose Query > New Query, and then enter the name of the list of values type that has a problem in the LOV Type property.

- 5 Press Enter to execute the query.
 - 6 For all the columns displayed, make sure all of them are LOV BOUNDED = Y.
 - 7 Set the Translation Table Name for all the columns displayed to S_LST_OF_VAL.
 - 8 Run MLOV Upgrade Utility in validation mode to make sure that there are no more errors.
- **LOV Domains Not in the S_LST_OF_VAL Table.** The message that appears in the log file for LOV domains that are not represented in S_LST_OF_VAL table is the following:

The following Validation checks for:

```
LOV domains in the repository that are not represented in  
S_LST_OF_VAL
```

This message means that an LOV domain is in the repository, but is not represented as a value in the list of values table, with a list of values type of LOV_TYPE. This can happen when you delete a record in the list of values table, instead of deactivating it, or when you enter an incorrect entry in the LOV Type property for a column added using a database extension.

For more information, refer to [“Deleting Compared to Deactivating Records” on page 685](#).

To fix this problem, add the LOV domain in the List of Values Administration view and specify LOV_TYPE in the Type list column, or correct the entry in the LOV Type property in the repository. See [“Adding Records” on page 685](#) for more information.

For any values found in the target tables without matching records in the list of values table, the script will create a matching record in the list of values table. These records are marked as inactive. Remember to add language-specific entries for these base records, so that they display in the active language.

Recompiling and Deploying

Every time you change the configuration to enable another column to be multilingual, you must compile a new .srf file based on the newly configured repository. Only the Newtable project needs to be compiled again. Additionally, you need to deploy the changes to users so that users can see the configured picklists in the desired language.

Integration Considerations

Enabling MLOVs does not affect just the Siebel eBusiness Applications client and the relevant target tables. Other features in your Siebel eBusiness Applications implementation must also consider this new configuration.

Enterprise Integration Manager

With Enterprise Integration Manager (EIM), you can import and export data. You can import data into both the list of values table and other tables in Siebel eBusiness Applications.

When importing data into the list of values table, the source table must have a language code and a name-value pair. This pair consists of the Display Value and the Language Independent Code.

When importing data into any other table, you must provide a language code for the /LANGUAGE command-line parameter for EIM. The source table must include the display value for multilingual columns in the language specified in the parameter. EIM validates imported data against list of values entries. The incoming data will be converted to associated language-independent codes during the import.

List of values entries that are marked inactive are ignored during the validation of multilingual LOV values during import by EIM.

When exporting data, you must specify a language code for the /LANGUAGE parameter, so that EIM can correctly translate the language-independent code in the table to the display value during the export.

For more information about command-line parameters for EIM, and information on EIM in general, see *Siebel Enterprise Integration Manager Administration Guide*.

Configuration Considerations

MLOVs are implemented below the business component level, so there are no special configuration considerations, other than what is described here. Fields that point to MLOVs with enabled target columns will automatically return display values that match the client language setting.

Since MLOVs are configured on a column basis, target columns that are not configured to be multilingual will behave as before; that is, target columns will store display values instead of language-independent codes.

For display, the underlying language-independent code is converted to its corresponding display value using a Siebel eBusiness Applications lookup. For searching and sorting, however, a database join is performed by your Siebel application to the list of values table. Therefore, when configuring the application, make sure that any configuration directly involving the list of values table is compatible with your Siebel application MLOV functionality.

It is only possible to have one multilingual picklist type running off each column. This means that for a table that has more than one business component mapped to it and hence several fields mapped to the same column, it is not possible to have a multilingual LOV attached to only one of the fields. This is checked by the MLOV upgrade utility running in validation mode. See [“Upgrading Existing Data Using the MLOV Upgrade Utility” on page 665](#).

MLOV Configuration and Coding Guidelines

The following guidelines should be followed when MLOVs are enabled in your environment.

- **LookupName and LookupValue functions.** These functions can only be used in calculated fields or search specification expressions. They cannot be used with Siebel scripting.
- **Pre/Post default values for fields with LOV picklists.** Always use the LookupValue function with *Expr:* in front of it. The first argument is the LOV Type and the second is the LIC. The function returns the language-specific Display Value. For example:

```
Expr: "LookupValue ("FS_PROD_ALLOC_RULES", "Default")"
```

- **Dynamic drilldowns and toggle applets.** These are usually based on a field that has a LOV value. For example, a dynamic drilldown might navigate the user to a Credit Card screen if the account type is equal to *Credit Card* or to a Savings screen if the account type is equal to *Savings*. Do not hard-code the drilldown or toggle conditions. Rather, use the LookupValue function (as described in the previous bullet).
- **Search specs for business components, links, applets, and picklists.** Always use the LookupValue function. For example:

```
[Invoice Code] = LookupValue('FS_INVOICE_CODE', 'Auction')
```

- **VB functionality.** VB does not offer a function to retrieve the language-specific Display Value. However, the Display Value must never be hard-coded; you should use the language-independent code instead. To write VB code using the language-independent code only, you must create calculated fields that hold the language-specific translation for a language-independent code.
- **Language and ResourceLanguage parameters.** Set these parameters *only* in the configuration file, for example, Language = < lang >, ResourceLanguage = ENU. If you do not set these parameters only in the configuration file, for example, when /L= < language > and ResourceLanguage = ENU, you will intermittently receive error 2009.

Querying and Multilingual Lists of Values

To run queries against fields that are controlled by MLOVs, use the Display Value for the search specification; do not use the Language Independent Code for querying. Querying will translate the search specification to the appropriate Language Independent Code to perform the query.

The Display Value used as the search specification should correspond to the Language being used by the application performing the query. If the query is being run through one of the Siebel interfaces (such as CORBA or COM), then the Language used for this translation is specified in the .cfg file used with the interface.

There is no difference to the user in the apparent functionality of the product when MLOVs are on or off. Internally, searches are applied using a function applied to the language-independent code. You can also do this with predefined queries and search expressions in the repository by using the same function (LookupValue (LOV Type, Language-Independent Code)). For more information about the LookupValue function and how it is used with MLOVs, see [“MLOV Configuration and Coding Guidelines” on page 675](#).

For more information about Query Operators and Expressions, see *Siebel Developer's Reference*.

Configuring Siebel Business Process Designer to Use MLOV-Enabled Fields

Additional configuration is required to enable the Siebel Business Process Designer to use MLOV-enabled fields. Siebel Business Process Designer compares values in target tables with values in the Business Process administration tables to determine whether a particular condition is true. For columns enabled for MLOVs, the value stored in the target table is the Language-Independent Code rather than the Display Value. However, the value in the Business Process Administration table is the Display Value. Siebel Business Process Designer cannot evaluate a condition by comparing the Language-Independent Code to the Display Value.

To enable Siebel Business Process Designer to work with MLOV-enabled columns, you must configure Workflow entities so that they compare the language-independent code in the target table with the language-independent code in the Business Process Designer administration table. You must do this for the following entities:

- Policy Conditions
- Action Arguments

Policy Conditions

Before you enable Policy Conditions, you must:

- Determine all the business component fields that are enabled for MLOVs.
- Of the fields that are MLOV-enabled, identify which ones are referenced by Policy conditions.

For each of the fields that reference a Workflow Policy condition, you must complete the following tasks:

- Create a new picklist to display LIC values.
- Create a new applet to display LIC values.
- Configure the Workflow Policy Column to use the new picklist and applet.
- Repick the values for existing workflow policies.

To create a LIC picklist for a Workflow Policy Column

- 1** In Siebel Tools, navigate to the Workflow Policy Column object type that you want to enable to use with MLOVs.
- 2** Find the Workflow Policy Column that references the MLOV enabled field.
- 3** In the PickList property field, click the picklist name.
The Pick Lists window appears in the Object List Editor.
- 4** Create a new picklist by copying the existing one and append LIC to the name.
For example, Picklist Account Status LIC.
- 5** Verify that the Sort Specification property of the picklist is set to "Name".

To create a new LIC applet for a Workflow Policy Column

- 1** Navigate back to the Workflow Policy Column selected in the previous procedure.

- 2** In the Applet property field, double-click the name of the associated applet.
The Applet window appears in the Object List Editor.
- 3** Create a new applet by copying the existing one and append LIC to the name.
- 4** Add a new list column to the applet for the language-independent Code.
 - a** In the Object List Editor, select List object type and then select the List Column object type.
 - b** In the List Column window, create a new record by copying an existing one, and then set the Field property to “Name.”

To configure the Workflow Policy Column

- 1** Navigate back to the Workflow Policy Column selected in the previous section.
- 2** In the PickList property field, select the new picklist created in [Step 4](#) of the procedure “[To create a LIC picklist for a Workflow Policy Column](#)” on [page 677](#).
- 3** In the Source Field property, change the value from Value to Name.
- 4** Compile changes.

To repick the Values

- 1** Log on using a client connected to the modified repository file.
- 2** From the application level menu, choose View > Site Map > Business Process Administration > Workflow Policies.
- 3** Repick the Values by selecting the conditions and reselecting the appropriate display values.

This will store the language-independent code.

Action Arguments

Before you enable Action Arguments, you must:

- Determine all the business component fields that are enabled for MLOVs.
- Of the fields that are MLOV enabled, identify which ones are referenced by Policy conditions.

For each of the fields that reference a Action Argument, you must complete the following tasks:

- Create a new picklist to display LIC values.
- Create a new applet to display LIC values.
- Configure the Action Arguments to use the new picklist and applet.
- Repick the values for the existing work flow policies.

To create a LIC picklist for a Workflow Policy Program Argument

- 1 In Siebel Tools, navigate to the Workflow Policy Program object type and the Workflow Policy Program that contains the argument that you want to enable for use with MLOVs.
- 2 Select the Workflow Policy Program Argument object type (child of Workflow Policy Program) and then select the Argument you want to enable for use with MLOVs.
- 3 In the PickList property field, click the picklist name.
The PickLists window appears in the Object List Editor.
- 4 Create a new picklist by copying the existing one and append LIC to the name.

To create a new LIC applet for a Workflow Policy Program Argument

- 1 Navigate back to the Workflow Policy Program Argument selected in the previous procedure.
- 2 In the Applet property field, double-click the name of the associated applet.
If no applet exists, you must create one.
The Applet window appears in the Object List Editor.
- 3 Create a new applet by copying the existing one and append LIC to the name.
- 4 Add a new list column to the applet for the language-independent code.
 - a In the Object List Editor, select List object type and then select the List Column object type.

- b** In the List Column window, create a new record by copying an existing one and set the Field property to Name.

To configure the Workflow Policy Program Argument

- 1** Navigate back to the Workflow Policy Program Argument selected in the previous section.
- 2** In the PickList property field, select the new picklist created in [Step 4](#) of the procedure “[To create a LIC picklist for a Workflow Policy Program Argument](#)” on [page 679](#).
- 3** In the Source Field property, change the value from Value to Name.
- 4** Compile changes.

To repick the values

- 1** Log on using a client connected to the modified repository file.
- 2** From the application level menu, choose View > Site Map > Business Process Administration > Workflow Policies.
- 3** Repick the Values of arguments for existing workflow policies.

For more information about Siebel Business Process Designer, see *Siebel Business Process Designer Administration Guide*.

Configuring Siebel Assignment Manager to Use MLOV-Enabled Fields

Additional configuration is required to enable Siebel Assignment Manager to use MLOV-enabled fields. Siebel Assignment Manager compares values in target tables with values in Assignment Manager administration tables to determine whether a particular condition is true. For columns enabled for MLOVs, the value stored in the target table is the Language-Independent Code rather than the Display Value. However, the value in the Assignment Manager administration table is the Display Value. Assignment Manager cannot evaluate a condition by comparing the Language-Independent Code to the Display Value.

To enable Siebel Assignment Manager to work with MLOV-enabled columns, you must configure Assignment Manager entities so that they compare the language-independent code in the target table with the language-independent code in the Assignment Manager administration table. You must do this for the following entities:

- Criteria Values
- Criteria Skills
- Workload Rules

Criteria Values and Criteria Skills

Before configuring Criteria Values and Criteria Skills, you must:

- Determine all the business component fields that are enabled for MLOVs
- Of the fields that are MLOV enabled, identify which ones are referenced by Criteria Values or Criteria Skills.

For each of the fields that reference Criteria Values or Criteria Skills (Assignment Attributes), you must set the Translate column to True and define the language-independent code field as the Translate Pick Field.

To configure Assignment Criteria and Skills for MLOVs

- 1 In the Object Explorer, select the Assignment Attribute object type.
- 2 In the Object List Editor, select the Assignment Attribute that you want to work with MLOV enabled fields.

- 3 Set the Translate property for the Assignment Attribute to True.
- 4 Set the Translate Pick Field property to the field name that stores the language-independent code.

Typically the Name field stores the language-independent code.

- 5 Compile changes.

Workload Rules

Before configuring Workload Rules, you must:

- Determine all the business component fields that are enabled for MLOVs.
- Of the fields that are MLOV enabled, identify which ones are referenced by Workload Rules.

For each of the fields that reference Workload rules, you must complete the following tasks:

- Create a new picklist to display LIC values.
- Create a new applet to display LIC values.
- Configure the Workflow Policy Column to use the new picklist and applet.
- Repick the values for existing records.

The detailed steps for completing these tasks are the same as the steps for configuring Workflow Policy Columns covered in [“Policy Conditions” on page 677](#).

For more information about Siebel Assignment Manager, see *Siebel Assignment Manager Administration Guide*.

Configuring Siebel Anywhere for Use with MLOV-Enabled Fields

Siebel Anywhere requires additional configuration to be able to use fields enabled for MLOVs.

To configure Siebel Anywhere for MLOVs

- 1** Open Siebel Tools.
- 2** In the Object Explorer, select the Table object type.
- 3** In the Tables window, query for S_UPG_KIT.
- 4** In the Object Explorer, select Column object type (child of Table).
- 5** In the Column window, select the STATUS column.
- 6** In the Translation Table Name field, click the drop-down list and select the S_LST_OF_VAL.
- 7** Compile an .srf file.

This enables Siebel Anywhere to use MLOV enabled fields.

After completing this procedure, you can perform standard tasks associated with Siebel Anywhere, such as creating a client repository upgrade kit and distributing to clients.

- Distribute the upgrade kit to Mobile Web Clients.
- Upgrade Siebel Servers with the new .srf file.

NOTE: Distributing Siebel executable to multilingual remote clients requires additional configuration. See *Siebel Anywhere Guide*.

Administering the Multilingual List of Values

After you have configured your Siebel eBusiness Application for MLOVs, use the List of Values views to administer and monitor LOV entries.

For detailed information about administering Lists of Values, see *Applications Administration Guide*.

Important Fields in List of Values Administration Views

Several list columns in the list of values views help you to administer multilingual LOVs:

- **Multilingual.** This field indicates which list of values types have been configured to be multilingual. When you run the MLOV upgrade utility (mlovupgd.exe), it sets this flag for the list of values entries. For more information on the upgrade script, refer to [“Upgrading Existing Data Using the MLOV Upgrade Utility” on page 665](#).

If you add entries after the script has been executed, you must manually update this information to reflect your configuration.

- **Language.** This field indicates for which language the entry is valid. The entries for this picklist come from the Language Administration view. To access this view, from the application-level menu, choose View > Site Map > Application Administration > Languages.
- **Translate.** This field indicates whether the entry’s display value can be changed—for instance, translated to another language.

Only the LOV types that are marked as translatable are candidates for multilingual configuration. For any entries added, you must update this information manually to reflect your configuration. Do not change existing Siebel entries. Doing so will not allow the LOV to be translated.

- **Language-Independent Code (or Code in the Explorer view).** This field is the internal code used for a list of values entry. It is stored in the database when MLOV is enabled and referenced by configurations. The language-independent code must be 30 characters or less. It is typically the English-American version of a particular selection value. The language-independent code cannot be changed.

- **Display Value.** This field is required and holds the text that will appear in picklists. The display value is stored in the database when MLOV is not enabled.

If there are display values for more than one language for a list of values entry, the display value shown is determined by the current active language.

Adding Records

When you add a new list of values record for a LOV type that has been multilingual-enabled, you also need to add records for all supported languages. For example, when adding a new entry for FREIGHT-TERMS type, you need to add values for all supported languages.

If you add a new entry and do not add corresponding additional records for each supported language, the language-independent code will be displayed instead of the display value when a user with one of these languages tries to view the information.

Adding records for all the languages you support is also important for Assignment Manager. For more information, refer to [“Configuration Considerations” on page 674](#).

For more information about adding records to the LOV table, see *Applications Administration Guide*.

Deleting Compared to Deactivating Records

As you administer MLOVs, you may find that there are records that you no longer need and would like to make inactive. If you delete a MLOV record, then records in other tables that have already been entered using that list of values record will no longer display correctly. The display value in the list of values entries is used to display the language-specific text.

Instead of deleting a record, inactivate it. Inactive values that have already been used and are referenced in other tables in the database will still display correctly. Inactive records are not included in any picklists, however, and are ignored by EIM when it performs validation against LOVs.

The Active list column in the List of Values Administration view is checked by default. To deactivate a record, remove the check mark.

If you try to delete a record, you will get a message asking you if you really want to delete the record, or just deactivate it. If you choose Inactivate from the dialog box, the check mark in the Active list column is removed.

If you do delete an LOV record, any language-independent codes in the target columns referring to the deleted record will then display the language-independent code. Searching and sorting will not function correctly on these values.

Constraints

LOVs are constrained in the following ways:

- You cannot have children of a list of values entry active when the parent list of values entry is inactive.
- Hierarchical LOVs do not support MLOVs.
- LOV_TYPE should not be enabled for multilingual list of values. It has a single-language entry only.

Multi-Value Group and Association Applets **12**

This chapter explains how to implement multi-value group and association applets in Siebel applications.

Multi-Value Group Applets

A multi-value group (MVG) applet (shown in Figure 178) is a dialog box that provides the means to display and maintain a set of records of data from another business component associated with the currently displayed business component record. The multi-value group applet is invoked from a control or list column in the originating applet.

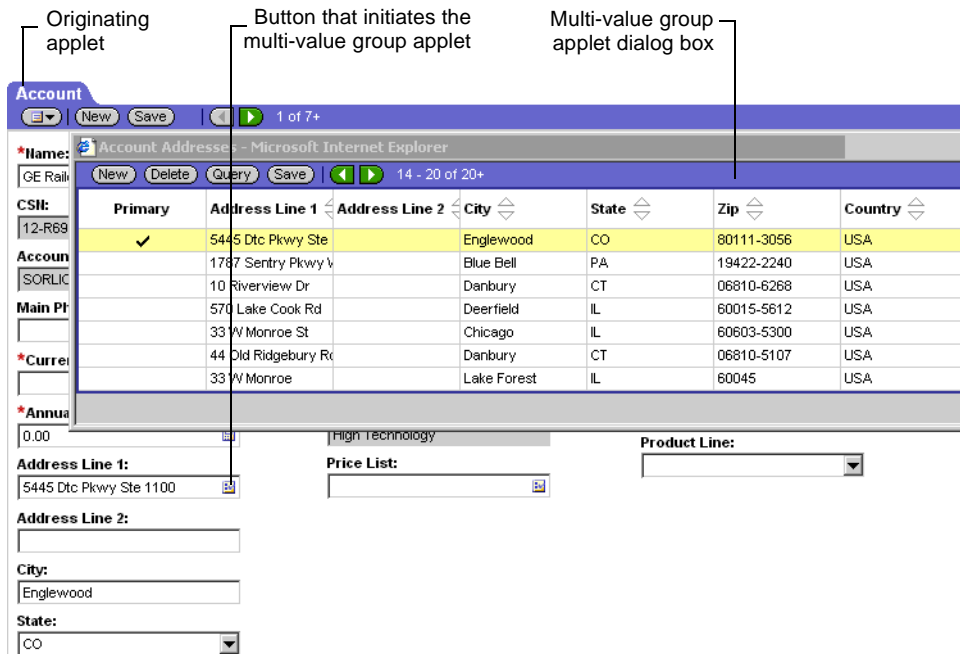


Figure 178. Multi-Value Group Applet in a Siebel Application

Figure 178 shows the Account Addresses multi-value group applet. It is invoked when the user clicks the check mark button to the right of the Address Line 1 text box. This multi-value group applet lists one or more addresses for the account. The record with a check mark in the Primary check box is the one whose data appears in corresponding controls in the originating applet.

While the multi-value group applet is open, the user can view the entire list of team member records for this account, not just the primary one. The user can also add, query, and delete records in this window.

Multi-value group applets are implemented using object types illustrated in Figure 179.

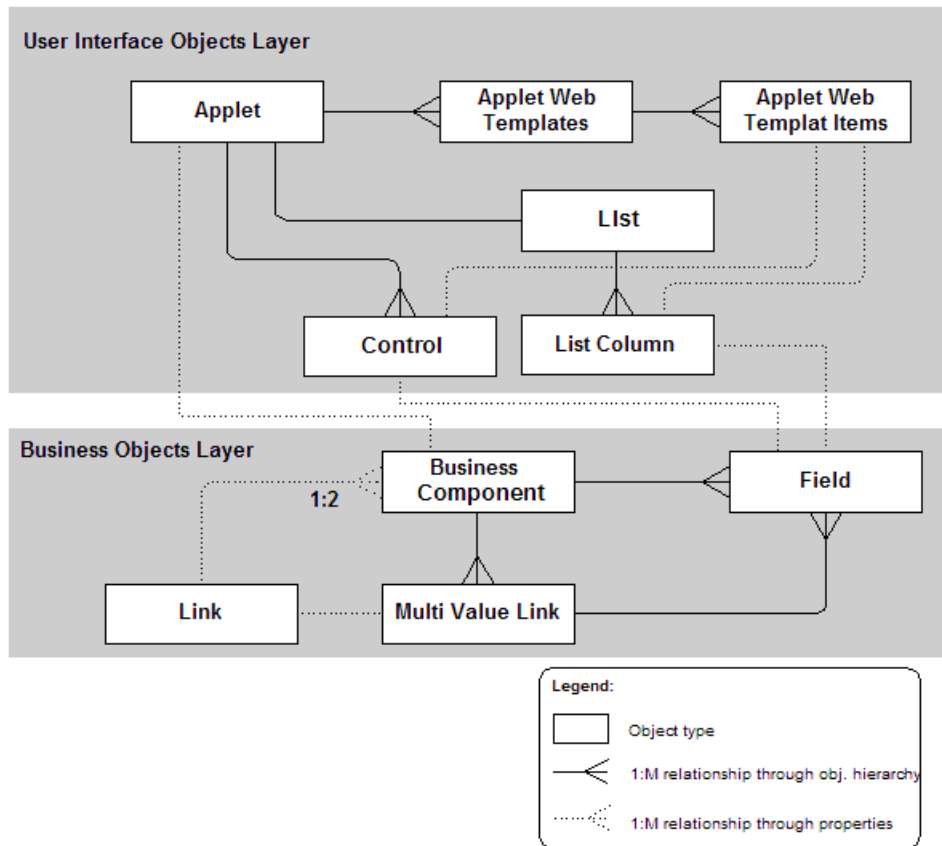


Figure 179. Multi-Value Group Architecture

Figure 180 shows the object definitions used in the implementation of a multi-value group applet in greater detail, and identifies their interrelationships.

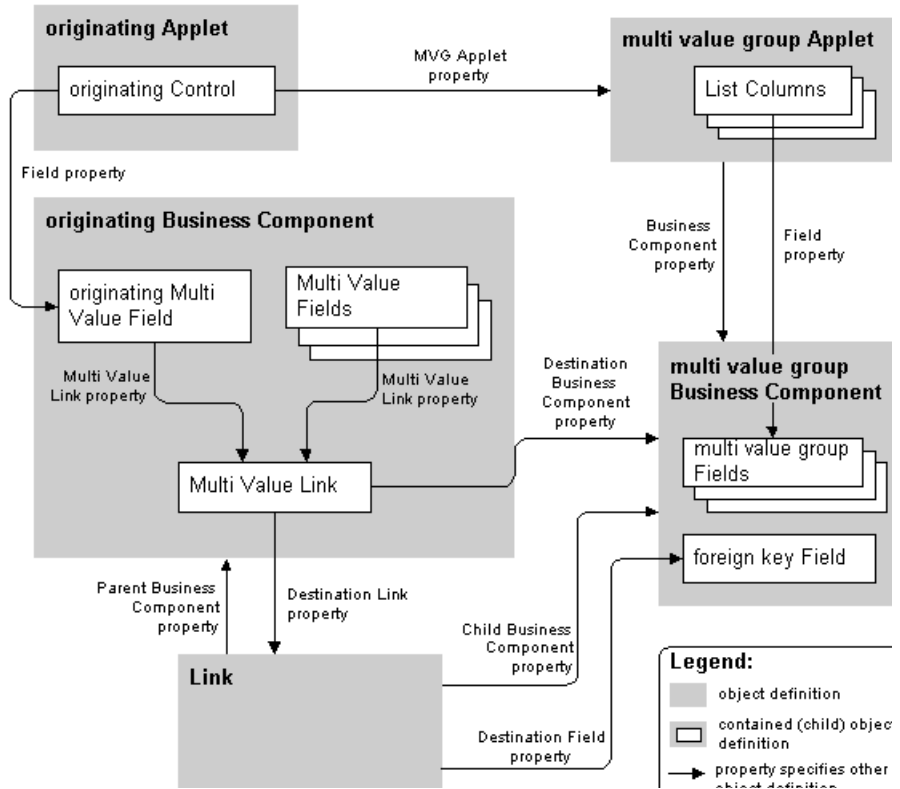


Figure 180. Multi-Value Group Details

The roles of the object definitions in Figure 180 are summarized in the following list and discussed in greater detail in the subsequent sections. The multi-value group example refers to the Account Address MVG applet illustrated in Figure 178 on page 688.

Each of the following objects is discussed in greater detail in the following sections.

- **Originating applet.** Contains the control or list column that invokes the multi-value group applet. In the example, the originating applet is called Account Entry applet.
- **Originating business component.** Business component of the originating applet. This business component (in the example, the Account business component) supplies the data presented in the originating applet (Account Entry applet).
- **Multi-value fields.** Fields in the originating business component that are populated by the multi-value link. Data population relationship is indicated by the presence of the multi-value link's name in their multi-value link property. The Field property in each identifies the corresponding field in the multi-value group business component that provides its data. Multi-value fields used in this multi-value group situation are Street Address, Address Id, City, Country, Fax Number, Postal Code, and State.

NOTE: If the field is a multi-value field, then the Required attribute will be ignored. In this case you can use a script in Siebel VB or Siebel eScript or the primary address field, if it exists.

- **Multi-value links.** Child of the originating business component. It identifies the link that provides the field values from the multi-value group business component. In the example, the multi-value link is Business Address.
- **Links.** Specifies the master-detail relationship between the originating and multi-value group business components. This is a property of the Multi Value Link object definition from which the fields in the originating business component obtain their values. In the example, the link is Account/Business Address.
- **Multi-value group applet.** Dialog box that appears when the user clicks on the ellipsis button in the originating applet. It lists the multi-value group business component records that are detail records in the master-detail relationship with the current originating business component record. It also provides the means to add, edit, and delete detail records. In the example, the multi-value group applet is called Account Address MVG Applet.

- **Multi-value group business component.** Stores the detail records of the master-detail relationship with the originating business component. The records displayed in the multi-value group applet are those in the multi-value group business component. In the example, the multi-value group business component is called Business Address.

Configuring the Originating Applet

The originating applet contains the control or list column that invokes the multi-value group applet. The originating applet has the following important property:

- **Business Component.** Identifies the originating business component.

The originating Control object or List Column object has the following important properties:

- **Field.** Identifies the originating field in the originating business component.
- **MVG Applet.** Name of the multi-value group applet to be invoked.
- **Runtime.** Must be set to TRUE.

Configuring the Originating Business Component

The originating business component is the business component of the originating applet. The data values that appear in the originating field and other multi-value fields are obtained from corresponding fields in a record in the multi-value group business component. The record from which these values are obtained is the one indicated as primary.

The originating business component has no essential properties for the configuration of a multi-value group. However, the field and Multi Value Link child object definitions are significant.

The originating field is the field specified in the Field property of the originating control or list column. Other than its relationship with the originating control, its role is identical to that of the other multi-value fields sharing the multi-value link. (A multi-value field is a field with a non-null Multi Value Link property.) Each of the multi-value fields participating in the multi-value group has the name of the multi-value link in its Multi Value Link property. The multi-value fields in the originating business component have the following important properties:

- **Multi Value Link.** Identifies the multi-value link that provides values, by way of the link object definition, from the multi-value group business component.
- **Field.** Identifies the field in the multi-value group business component that, by way of the multi-value link and link object definitions, provides values for the field in the originating business component.

Pick maps can be used for multi-value fields similarly to how they are used for single-value fields. The MVF Pick Map object is a child object of Multi Value Field. Each pick map defines a correspondence between a field in the multi-value group business component and one in the originating business component. These correspondences provide the information required to immediately update the current originating business component record with information from the MVG business component when a record is picked.

Each MVF Pick Map object definition has two important properties:

- **Field.** Identifies a field in the originating business component that is to be populated by data from a field in the MVG business component, when the PickRecord method is invoked.
- **Pick List Field.** Identifies a field in the MVG business component that is the source of data for the field in the Field property of the Pick Map object.

An example is the MVF pick map on the State multi-value field of the Account business component. Account has a multi-value link to the Business Address business component, where it obtains address information.

Configuring the Multi-Value Link

The multi-value link object definition is a child of the originating business component. It identifies the link that provides the field values from the multi-value group business component. The multi-value link in the originating or joined (in the case of an indirect multi-value link) business component has the following important properties:

- **Destination Link.** Identifies the link object definition that provides the master-detail relationship between the originating (or joined) and multi-value group business components.

- **Primary Id Field.** Identifies the foreign key field in the originating, or joined, business component. The foreign key field identifies the primary record in the set of records for one multi-value group (in the multi-value group business component). The primary record is the one that displays in the originating or employing business component.
- **Destination Business Component.** Name of the child business component.

NOTE: An indirect multi-value link may be used in place of a conventional multi-value link when there is an existing link object definition which would be appropriate for use in a multi-value link, but the originating business component is different from the master business component. If there is a Join object definition that joins the desired master business component to the master business component of the link, the existing link can be used in the multi-value link.

When configuring a pick applet invoked from a multi-value group applet, define the Pick List on the originating field in the originating business component, not on fields in the multi-value group business component. Changes must also be made to the pick map. See the following example of configuring conjoined fields in MVG applets.

On the Contact Business Component, there is a Multi-Value Link, Contact Category (with Contact Category as the destination buscomp), and a Multi-Value Field, Category (with Category as a destination field). On the Contact Category BusComp, Category is a joined field with a picklist.

To modify Category using a multi-value group applet, you must configure the Category field on the Contact buscomp by completing the following steps:

- 1 Set Picklist = Picklist Contact Category
- 2 Add two new entries to Pickmap:
 - Field: Pick List Field
 - Category: Name
 - Categ Id: Id

- 3 Category Id is a new MVF created by setting the following:

MVL = Contact Category

Field = Category Id

Configuring Links

The Link object definition specifies the master-detail relationship between the originating and multi-value group business components. This makes possible the Link object definition from which the fields in the originating business component obtain their values. The Link object type has the following important properties:

- **Parent business component.** Identifies the originating business component.
- **Child business component.** Identifies the multi-value group business component.
- **Source Field.** Identifies the field in the originating business component that serves as a unique ID to that business component. If this property is blank, it indicates that the field that maps to the ROW_ID column, generally called Id, is the source field.
- **Destination Field.** Identifies the field in the multi-value group business component that identifies the master record for each detail record. It is a foreign key that points back to the originating business component.

NOTE: In a link based on an intersection table (the relationship between the originating and MVG business component is many-to-many—that is, one in which the Inter Table, Inter Parent Column and Inter Child Column properties are non-blank), you do not specify the Source Field or Destination Field properties. You can specify a source field (although it is not common to do so). Destination field always defaults to Id.

Configuring the Multi-Value Group Business Component

This business component stores the detail records of the master-detail relationship with the originating business component. The records displayed in the multi-value-group applet are from the multi-value group business component.

The multi-value group business component has no important properties with respect to its role in the implementation of a multi-value group. It has field child object definitions that are used in the following ways:

- To store data for a field in the multi-value group

Each field with this role is represented by a list column in the multi-value group applet. It may also participate in the multi-value link to supply data to a corresponding field in the originating business component.

- To identify the primary record in the multi-value group

Primary records are identified by the primary field which is specified in the Primary Field Id property in the multi-value link.

NOTE: For all intents and purposes, the primary field has nothing to do with the multi-value group business component. It does have relevance to the originating business component, the multi-value link, and the multi-value group applet.

- As the destination field of the link

The field with this role is a foreign key to the originating business component.

Using the MVG Wizard

You can use the MVG Wizard to help you create and define the necessary relationships between business components and define multi-value fields. If you need to make any modifications, you can use the Back button to return to the appropriate dialog box.

To configure a multi-value group using the MVG Wizard

- 1** Choose File > New Objects.

The New Object Wizards dialog box appears.

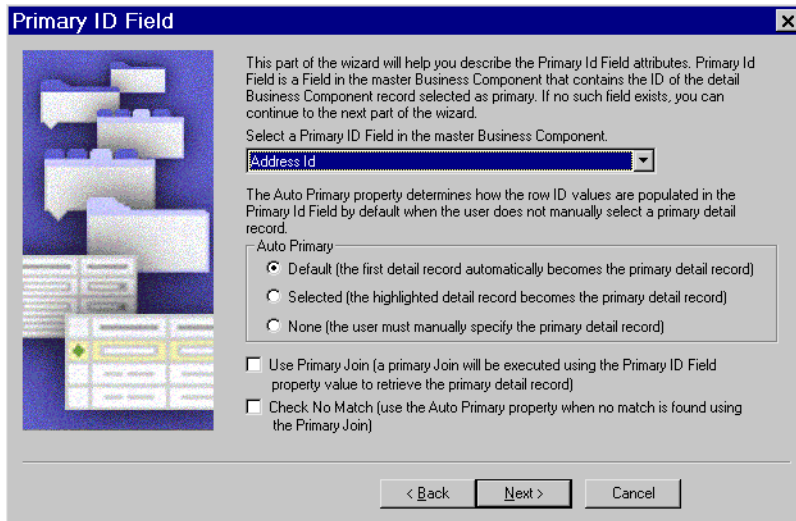
- 2** Under the General Tab, double-click the MVG icon.

The Multi Value Group dialog box appears.

- 3** In the Multi Value Group dialog box, select:

- The project to which the MVG will belong. Only locked projects are available for you to select.
 - The master business component. The business component must belong to the selected project.
 - Click Next.
- 4** In the second Multi Value Group dialog box, do the following:
- Select the detailed business component.
 - Enter a name for the Multi Value Link.
 - Click Next.
- 5** In the Direct Links dialog box, select the link that you want to use, and then click Next.
- The available links are those that already exist between the master and detail business component.
- 6** In the Primary ID Field, do the following:
- Select the Primary ID Field in the Master Business Component.
 - Set the value for the Auto Primary property.
 - Select other options as necessary.

- Click Next.



- 7 In the Multi Value Link dialog box, select the properties that you want to define and then click Next.
- 8 In the Multi Value Fields dialog box, enter information for creating multi-value fields on the master business component.
 - Select a field on destination business component.
 - Enter a name for the multi-value field.
 - Click Add.
 - Repeat for each field you want to add.
 - Click Next.
- 9 In the Finish dialog box, review the information you have entered for the MVG and then click Finish.

Configuring the Multi-Value Group Applet

This applet is the dialog box that appears when the user clicks the ellipsis button in the originating applet. It lists the multi-value group business component records that are detail records in the master-detail relationship with the current originating business component record. It also provides the means to add and delete detail records. The multi-value group applet contains list column object definitions that present the data from corresponding fields in the multi-value group business component.

The multi-value-group applet has the following important properties:

- **Business Component.** Identifies the multi-value group business component.
- **Class.** Enter a value of `CSSFrameList` in this property. This setting indicates that this is a standard list applet.
- **Type.** Enter a value of `MVG` in this property. This setting indicates that this is a multi-value group applet. This configures the behavior of the dialog box and button controls.
- **Title.** Identifies the multi-value group applet to appear in the title bar.

The List Column object definitions in the multi-value group applet have the following important property:

- **Field.** Identifies the field in the multi-value group business component from which the list column displays data.

If users will perform queries on fields mapped to MultiValue Groups, do the following.

Determine the query destination field (the multi-value link field being used for the query)—the field that is mapped to a database column.

In Siebel Tools, remove the check mark from the Use Default Sensitivity property for that destination field.

NOTE: For fields of type DTYPE_ID the follow rules hold true:

Queries are case-sensitive if Use DefaultSensitivity is TRUE and the .cfg file CaseInsensitive setting is FALSE.

Queries are case-insensitive if Use Default Sensitivity is TRUE and the .cfg file CaseInsensitive setting is TRUE.

Siebel data types are discussed in more detail in *Object Types Reference*.

Using the MVG Applet Wizard

You can use the MVG Applet Wizard to create an MVG applet. It will help you define all the necessary object definitions for an MVG applet.

To create and MVG applet

- 1** Choose File > New Object from the Siebel Tools main menu.
The New Object Wizard dialog box appears.
- 2** Click the Applets tab and then double-click the MVG Applet icon.
The General page of the MVG Applet Wizard appears.

- 3 In the General Page, enter the following information for the applet, and then click Next.

Field	Comment
Project	Only locked projects appear in the picklist.
Business Component	The business component that the applet is based on.
Applet Name	A unique name for the Applet.
Display Title	The name to appear in the user interface.

The wizard will use information to create an applet object and define the required applet properties.

- 4 In the Web Layout-General page, enter the Web templates to use for the applet, and then click next.

Typically, MVG applets use the Popup List template.

NOTE: If you plan to include a New button on your MVG applet, you will also need to define an Edit mode manually, using the Popup Query template.

For a complete description of templates, see *Siebel Developer's Reference*.

- 5 In the Web Layout - Fields page, select the fields that you want to appear on the applet, and then click Next.

The fields that appear in the Available pane are those fields defined for the business component that you selected in [Step 3 on page 701](#).

- 6 In the second Web Layout-Fields page, choose the controls in the Available Controls box that you want to appear on the applet, and then click Next.

All the entries in the Selected Controls box are added by default. If you wish to exclude some of the controls and move them to the Available Controls box, select the controls and click the activated arrow.

NOTE: The available controls come from the “Model HTML Controls” applet. This applet specifies the available controls and also to which template each control is mapped. Users can modify this applet if necessary by adding or removing controls from the applet.

- 7 Review the information displayed in the Finish page, and then click Finish.

The MVG Applet Wizard creates the applet and supporting object definitions based on the selections you made.

NOTE: You can return to previous pages by clicking the Back button.

Association Applets

An *association applet* (shown in [Figure 181](#)) provides users with the ability to associate a parent record with one or more children through an intersection table. For example, association applets are provided for assigning Team Members to an Account, Contacts to an Opportunity, and Products to a Price List.

An association applet is a dialog box. Multi-selection is only available in the association applet. Fields in association applets cannot be updated.

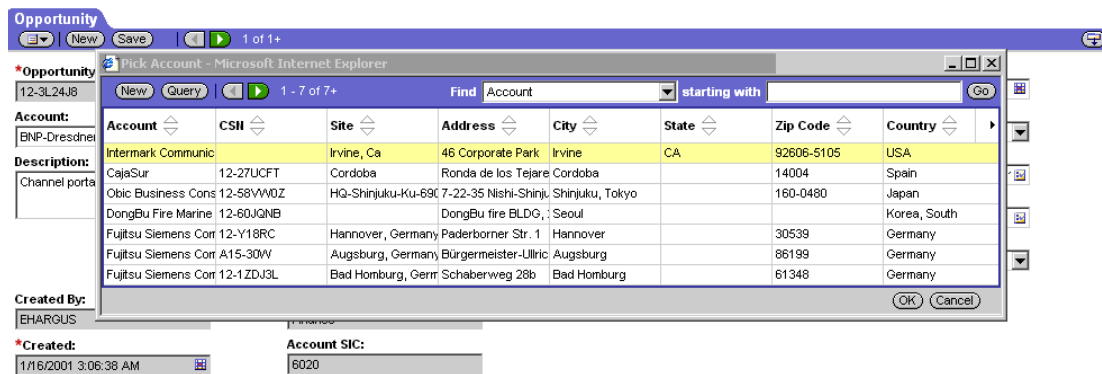


Figure 181. Association Applet in a Siebel Application

The association applet lists the records from a business component. The user selects one or more records with the aid of the Find and Starting With controls, if needed, and clicks the Add button to associate the selected record with the active master record.

Association applets are used only with pairs of business components that have a many-to-many relationship. A many-to-many relationship in Siebel applications is implemented by means of an intersection table and a pair of links.

When a one-to-many (simple master-detail) relationship exists between business components, an association applet is unnecessary, and records can be directly added or inserted in the applet displaying the detail business component. The means to add and delete records can be provided by using a master-detail view or a multi-value group applet. When a many-to-many relationship exists, an association applet provides the only means to associate a pair of records from their respective business components. [Figure 182 on page 704](#) and [Figure 183 on page 705](#) illustrate the reason for this.

NOTE: Records in an association applet are read-only; they cannot be modified from within the association applet dialog box.

Figure 182 illustrates how a one-to-many relationship is implemented.

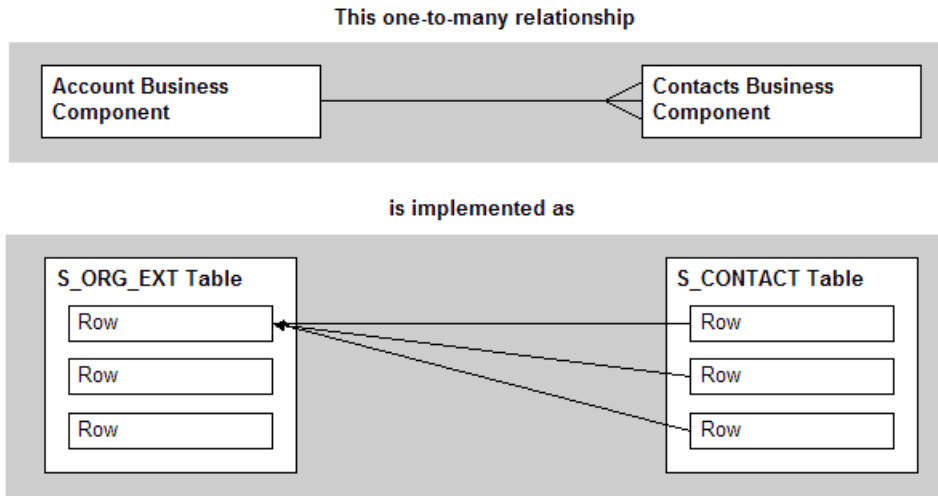


Figure 182. Row Relationships in a One-to-Many Relationship

The two applets in a master-detail view display one master record and a list of detail records in their respective business components. A foreign key in each of the detail records points to the one master record. To add another detail row, a row is added to the detail table (S_CONTACT in the illustration), and a value is set in the foreign key that points back to the master row. Every row in the detail table has a master row because of the link relationship between the master and detail business components. Adding a row to the detail table always results in the linkage of the new row to a row in the master table. No additional applet is necessary to create an association between a new detail row and a master row.

The situation in a many-to-many relationship is illustrated in [Figure 183](#).

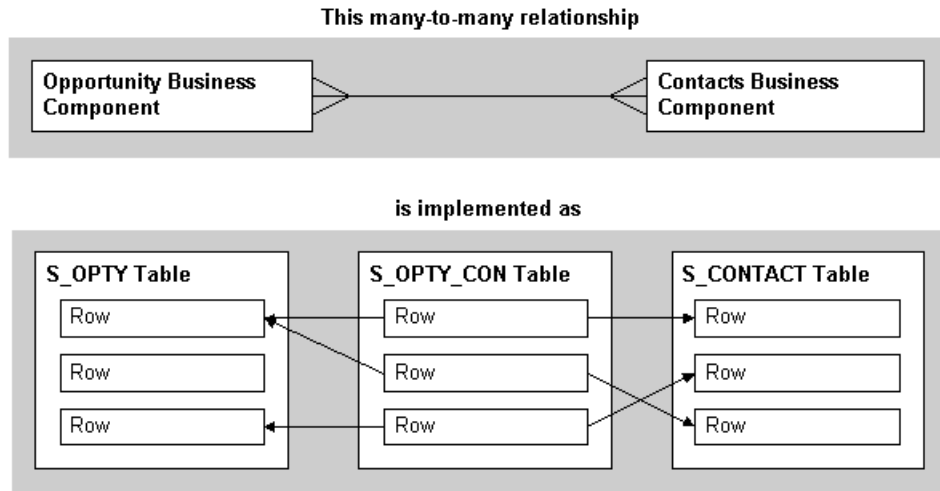


Figure 183. Row Relationships in a Many-to-Many Relationship

“Adding” a record to the detail business component in a many-to-many relationship can in reality mean associating an existing detail record to a master record, rather than creating a new detail record from scratch. This is because master and detail are relative terms in a many-to-many relationship. For example, the Opportunity and Contacts business components in [Figure 183](#) can be displayed as one opportunity to many contacts, or one contact to many opportunities, depending on the active view.

In this situation, the user needs to be presented with a selection list of available detail records. If users see the desired detail record in the selection list, they choose it. If not, they have the option to create a new detail record. In the context of a many-to-many relationship, creating a new association for an existing detail record is called association; creating a new detail record and an association is called addition. Both association and addition of detail records result in the creation of a new row in the intersection table. Addition also results in the creation of a new row in the detail table.

In the association applet, the Add button performs association and the New button performs addition.

In Siebel applications, association applets are invoked in one of two ways:

- From the list applet in a master-detail view, by choosing the Edit > Add New Record menu option.
- From a multi-value group applet window, by clicking the New button.

Each of these scenarios is discussed in the following sections.

NOTE: An association applet cannot be configured to be constrained or filtered through properties the way a pick applet can (using the Constrain property of a Pick List). To constrain an association applet, you must use Siebel VB or Siebel eScript to query using the Exists clause in the `WebApplet_Load` event on the association applet.

Association Applets Invoked from Master-Detail Views

Figure 184 illustrates an association applet invoked in a master-detail view by selecting New Record from the menu on the Contacts list applet.

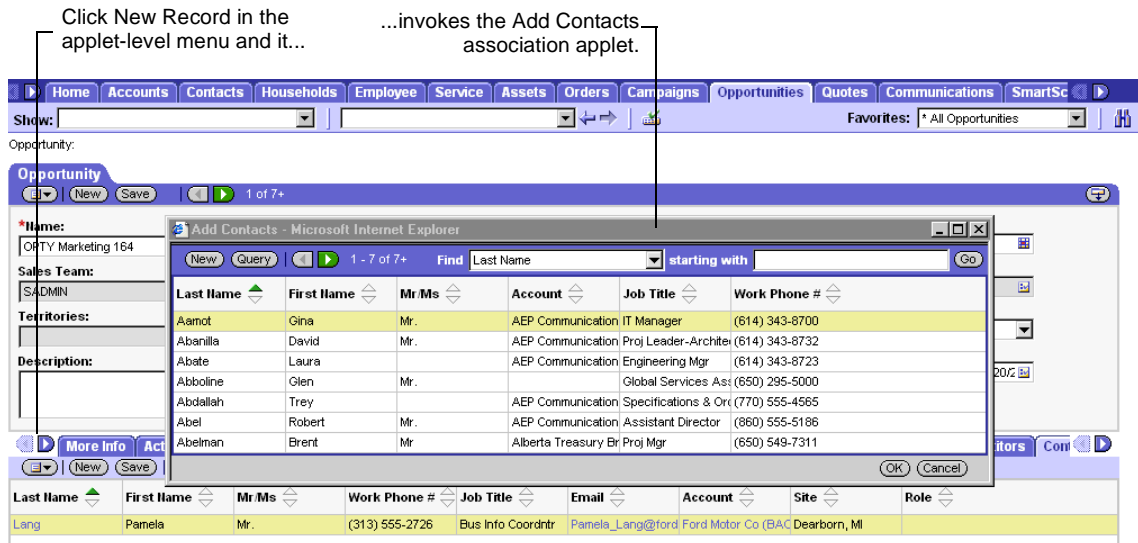


Figure 184. Association Applet Invoked from a Master-Detail View

The master-detail view in Figure 184 is Opportunity Detail - Contacts List View, one of two master-detail views displaying opportunity and contact information. The other is Contacts Detail - Opportunities List View, which displays the inverse master-detail relationship. When the user chooses New Record from the Opportunities edit menu, the Opportunities dialog box appears for selection of an existing opportunity record to insert, or for creation of a new opportunity record. A new opportunity record is created by clicking the New button and then entering data into the new record in the Add Opportunities dialog box.

For example, the Add Contacts dialog box is implemented as an association applet called Contact Assoc applet.

Association applets are implemented using the object types illustrated in [Figure 185](#).

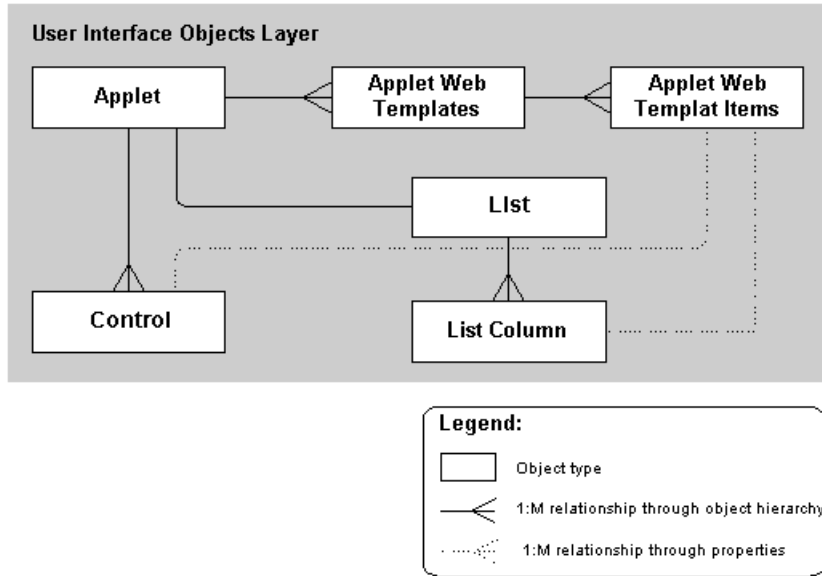


Figure 185. Association Applet Architecture

The details of the object relationships are shown in [Figure 186](#).

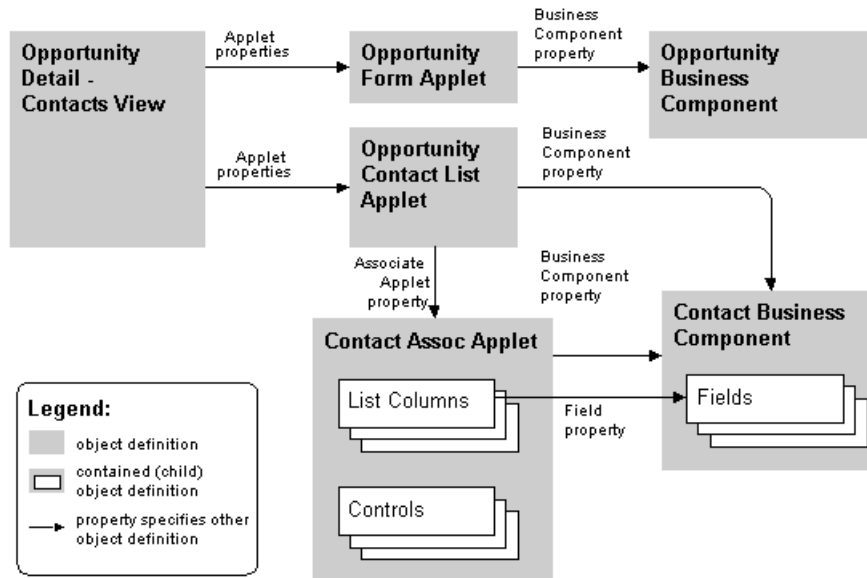


Figure 186. Association Applet (Invoked from Master-Detail View) Details

Roles of the object definitions in [Figure 186](#):

- **View (Opportunity Detail - Contacts List view).** Provides the context in which the association applet is invoked, although no properties of the view directly identify the association applet. The Business Object property of the view establishes the master-detail relationship between the business components whose data is displayed.
- **Master applet (Opportunity form applet).** Form applet that displays one record from the master business component. It has no special properties to configure.
- **Detail applet (Opportunity Contact list applet).** List applet that displays a list of records from the detail business component that are detail records for the current master record in the master business component. The name of the association applet is specified in its Associate Applet property.

- **Business components (Opportunity and Contact).** Provide the data for their respective applets in the view. The detail business component also provides the data displayed in the association applet.

NOTE: In the association applet, all records from the detail business component are displayed; in the detail applet, the only records displayed are those which have already been associated to the current master record.

- **Association Applet (Contact Assoc applet).** Implements the dialog box that appears when the user attempts to add or insert a record in the detail applet. It has a Type property value of Association List, which indicates that it is an association applet, and a Class property value of CSSFrameList, indicating that it is a list applet. The association applet is configured as a standard list applet, with a List child object definition that, in turn, has List Object child object definitions.
- **List columns.** Specify the fields that are displayed in the association applet, and in what order. They duplicate some or all of the list columns in the detail applet in the view.
- **Controls.** Several specialized controls appear in an association applet. They are the following:
 - **Check button.** Associates selected records to the current parent. The result at the table level is to create an intersection table row between the row identified in the master applet and the row identified in the association applet. The control is named `PopupQueryAdd` and has a method invoked of `AddRecord`.
 - **New button.** Creates an empty scrolling table row in the association applet for user entry of a new detail business component record. Following entry of the new record, it is inserted in the detail applet. The result at the table level is to create a new row in the detail table, and an intersection table row between the row identified in the master applet and the row created in the association applet. The control is named `ButtonNew` and has a method invoked of `NewRecord`.
 - **Cancel button.** Dismisses the dialog box.

- **Find combo box.** In combination with the Starting With text box and Find button, provides the user with search capabilities for locating the desired record in the association applet. The user selects in this combo box the field to search.
- **Starting With text box.** Text box where the user enters the search criteria. The criteria entered in this control are automatically completed by a wild card. It is not possible initiate another search and enter exact search criteria.
- **Go button.** The user clicks this button to initiate the search specified in the Find combo box and Starting With text box.

Association Applets Invoked from Multi-Value Group Applets

Figure 187 illustrates an association applet invoked from a multi-value group applet. Figure 188 on page 712 shows the dialog box that appears when you click the New button. Figure 189 on page 712 shows the dialog box that appears when you click the Add button.

Then click the New button in a multi-value group applet...

Click the Select button in an MVG field to invoke an MVG

Primary	Priority	User ID	Last Name	First Name	Position	Organization	Role	Credit Allocation	Manual
		TLOOMIS	Loomis	Timothy	Director, Sales Cont	Default Organization		100%	✓
	✓	EHARGUS	Hargus	Emily	District Manager-001	Default Organization		100%	✓

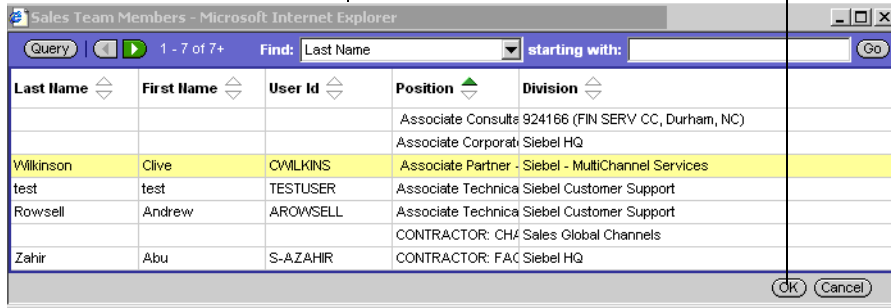
Figure 187. Multi-Value Group Applet Invoked from Organization Field

Multi-Value Group and Association Applets

Association Applets

After clicking New an association applet appears.

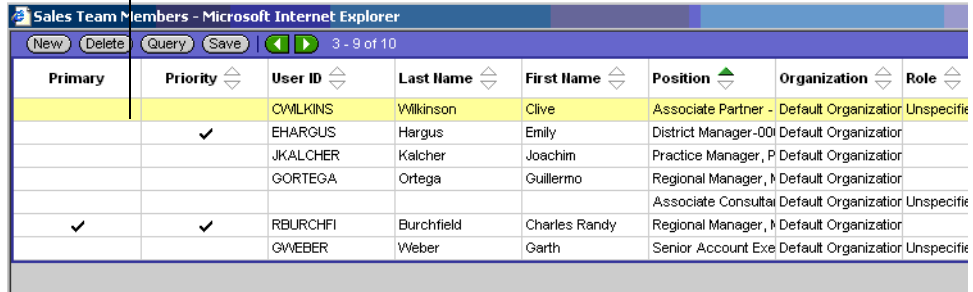
Click OK to add entry to list.



Last Name	First Name	User Id	Position	Division
			Associate Consult	924166 (FIN SERV CC, Durham, NC)
			Associate Corporat	Siebel HQ
Wilkinson	Clive	CWMLKINS	Associate Partner - Siebel - MultiChannel Services	
test	test	TESTUSER	Associate Technica	Siebel Customer Support
Rowsell	Andrew	AROWSELL	Associate Technica	Siebel Customer Support
			CONTRACTOR: CHA	Sales Global Channels
Zahir	Abu	S-AZAHIR	CONTRACTOR: FAC	Siebel HQ

Figure 188. Association Applet Invoked from a Multi-Value Group Applet

A new record is added to the list.



Primary	Priority	User ID	Last Name	First Name	Position	Organization	Role
		CWMLKINS	Wilkinson	Clive	Associate Partner - Default Organization		Unspecific
	✓	EHARGUS	Hargus	Emily	District Manager-001	Default Organization	
		JKALCHER	Kalcher	Joachim	Practice Manager, P	Default Organization	
		GORTEGA	Ortega	Guillermo	Regional Manager, R	Default Organization	
					Associate Consultat	Default Organization	Unspecific
✓	✓	RBURCHFI	Burchfield	Charles Randy	Regional Manager, R	Default Organization	
		GWEBER	Weber	Garth	Senior Account Exe	Default Organization	Unspecific

Figure 189. Added Entry

The applet in the upper section of the view shown in [Figure 187 on page 711](#) is an Account form applet, which is used in various views to display a single record of account information at the top of the view. Five of the fields in this applet display multi-value fields: Sales Team, Organizations, Territories, and Account. Each of these text boxes has a check mark button that invokes a specific multi-value group applet when clicked.

Some multi-value-group applets add and remove records without the use of an association applet. Such applets are based on a one-to-many relationship between the master and detail business components, and no intersection table is involved. The Account Address MVG applet (invoked from the Address text box in the Account form applet) is that kind of multi-value group applet. You can confirm that no association applet is involved by examining the value in the Associate Applet property in the multi-value group applet's object definition, which is blank in this case.

Other multi-value group applets are based on a many-to-many relationship implemented through an intersection table. The user may either create or associate records. This requires that an association applet be invoked when the New button in the multi-value group applet is clicked. The Industry Mfg applet, invoked from the Industries text box, is of this type. The name of an association applet (Industry Assoc applet) appears in the Associate Applet property in the multi-value group applet's object definition.

In [Figure 187 on page 711](#), the user has clicked the ellipsis button to the right of the Industries text box, and the Add Industries dialog box has appeared. The Add Industries dialog box (association applet) allows the user to add an existing industry record to the multi-value group, or to create a new industry record in the multi-value group. A new Industry record is created by clicking the New button, and entering data into the new record in the Add Industries dialog box.

The Add Industries dialog box is implemented as an association applet called Industry Assoc applet. The details of the object relationships are shown in Figure 190.

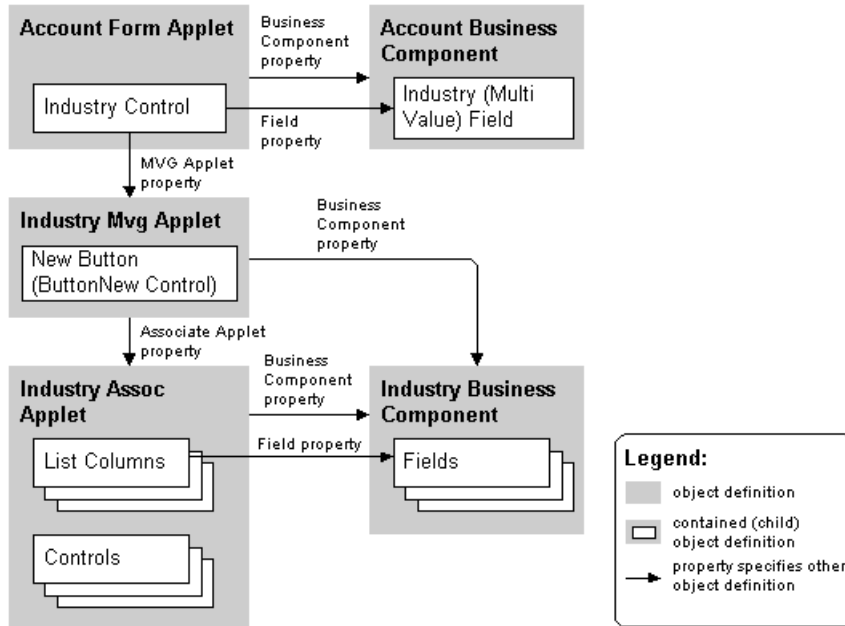


Figure 190. Association Applet (Invoked from Multi-Value Group Applet) Details

The roles of the object definitions in Figure 190 are the following:

- **Form applet (Account Form applet).** Contains one or more text box controls displaying multi-value fields. The MVG Applet property for each of these text box controls identifies a multi-value group applet that is invoked when the user clicks the ellipsis button to the right of the text box.
- **Multi-value group applet (Industry Mfg. applet).** Displays the list of records assigned to the multi-value field in the form applet. The Associate Applet property in the multi-value group applet's object definition identifies the association applet to invoke when the user clicks the New button in the multi-value group applet.

- **Multi-Value Group business component (Industry).** Stores the (detail) multi-value group records for each master business component record. The multi-value group business component supplies records to both the multi-value group applet and the association applet.
- **Association applet (Industry Assoc applet).** Implements the dialog box that appears when the user attempts to add or insert a record in the multi-value group applet. The association applet has a Type property value of Association List, which indicates that it is an association applet. It has a Class property value of CSSFrameList, indicating that it is a list applet. The association applet is configured as a standard list applet, with a List child object definition that has List Object child object definitions.

The child object definitions for the association applet are described in greater detail at the end of [“Association Applets Invoked from Master-Detail Views” on page 707](#). Typically Association applets are based on the same business component as multi-value group applets.

Multi-Value Group and Association Applets

Association Applets

Special-Purpose Applets and Controls **13**

This chapter discusses special-purpose applets and controls:

- Chart applets
- Tree applets
- File attachment applets
- Pop-up windows
- ActiveX controls
- HTML content controls
- Dynamic toggle applets

Chart Applets

A chart applet graphically displays data from a business component in various formats for analysis of trends, category comparison, and other data relationships. Any data in a business component can be included in a chart. The data in a chart applet reflects the current query for the business component. The user can update the chart with changes to the query by clicking inside the chart. [Figure 191](#) shows a chart applet in a view.

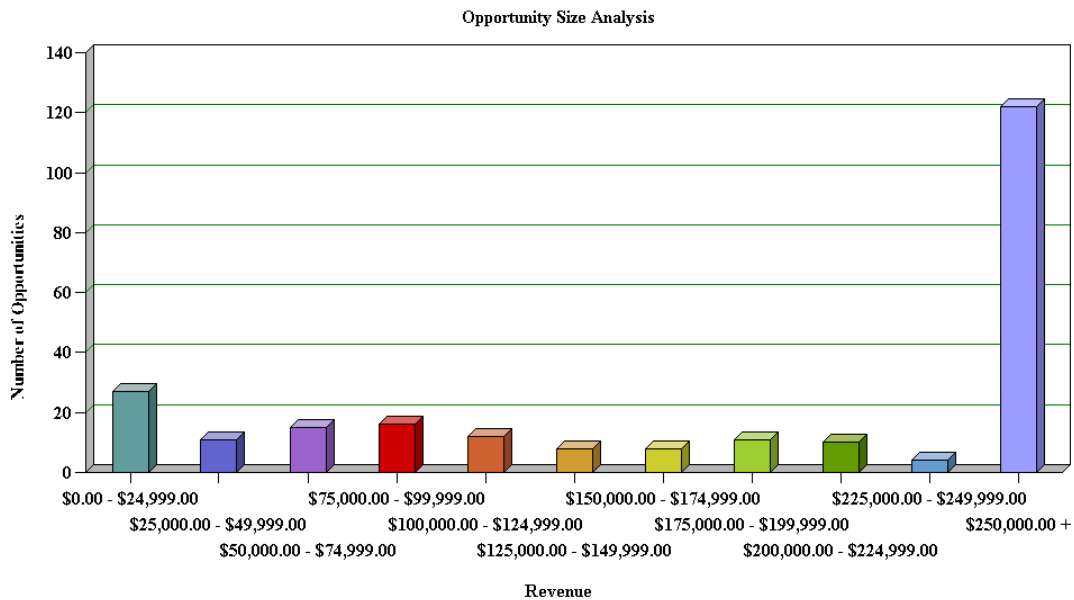


Figure 191. Opportunity Size Analysis View

This view, titled Opportunity Size Analysis (Oppty Chart View - Opportunity Size Analysis in Siebel Tools), lists all opportunities in the upper (list) applet and aggregates them by size in the lower (chart) applet. By default, the chart applet in this view (Oppty Chart Applet - Competitor Frequency Analysis) displays the data in bar chart format, in a specific type of bar chart called 3dBar. The user can select different chart types from the Type picklist at upper right in the chart applet. Chart types are discussed in [“Chart Layout Options” on page 722](#).

NOTE: To change the size of the legend for a chart applet, right-click on the legend and select one of the options.

Axis Terminology

Specialized terminology is used for axes in Siebel Tools and Siebel applications. Each axis has a special name, as shown in [Table 47](#).

Table 47. Axis Terminology

Axis	Name	Meaning in Bar Charts	Meaning in Line Charts	Meaning in Pie Charts
X axis	Category	The horizontal axis (except in horizontal bar charts, in which the X axis is vertical along the left).	The horizontal axis.	The set of pie slice labels.
Y axis	Data Values	The vertical axis (except in horizontal bar charts, in which the Y axis is horizontal along the bottom).	The vertical axis.	The percentage of the circle occupied by each pie slice, and the corresponding numeric value.
Z axis	Series	A set of labels in the legend. In the stacked bar or cluster bar charts, each series label corresponds to a bar segment or bar of a particular color appearing in each stack or cluster.	A set of labels in the legend. In line charts, each series label in the legend corresponds to one curve.	Do not use a series field with pie charts, because only the first entry in each series will be charted.

An example of a chart with all three axes is the Project Revenue Analysis chart shown in Figure 192.

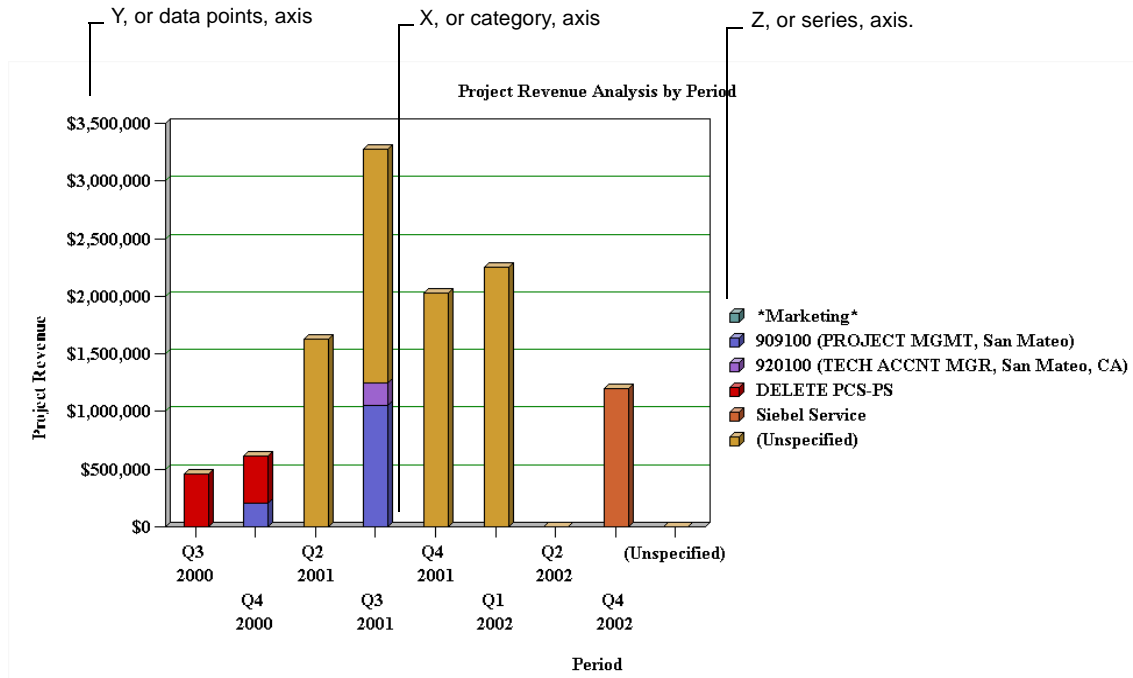


Figure 192. Project Revenue Analysis Chart in Siebel Service

In this chart, the amount of revenue is plotted on the Y (data values) axis, quarters appear on the X (category) axis, and each bar color (Z, or series, axis) identifies a different project.

NOTE: In charts with two Y axes, the first Y axis refers to the vertical axis on the left side, while the second Y axis refers to the one on the right side.

Chart Layout Options

The user can select different chart types from the Type picklist at the upper right in most chart applets. Chart types provide various layout options, including horizontal bar, stacked bar, pie, line, scatter, spline, and combo (combined line and bar). Several of these are available in either 2- or 3-dimensional format. The 3-dimensional types are functionally the same as the corresponding 2-dimensional types, but provide the illusion of bar, line, or pie thickness for visual attractiveness.

The following styles of charts are available (although not all styles are supported for all chart applets).

Bar Charts

Bar charts are typically used to compare the absolute difference in data from one category to another.

- **3dBar.** The 3dBar type divides data from the source records into categories, and displays the total for each category as a vertical bar. This is shown in [Figure 193](#).

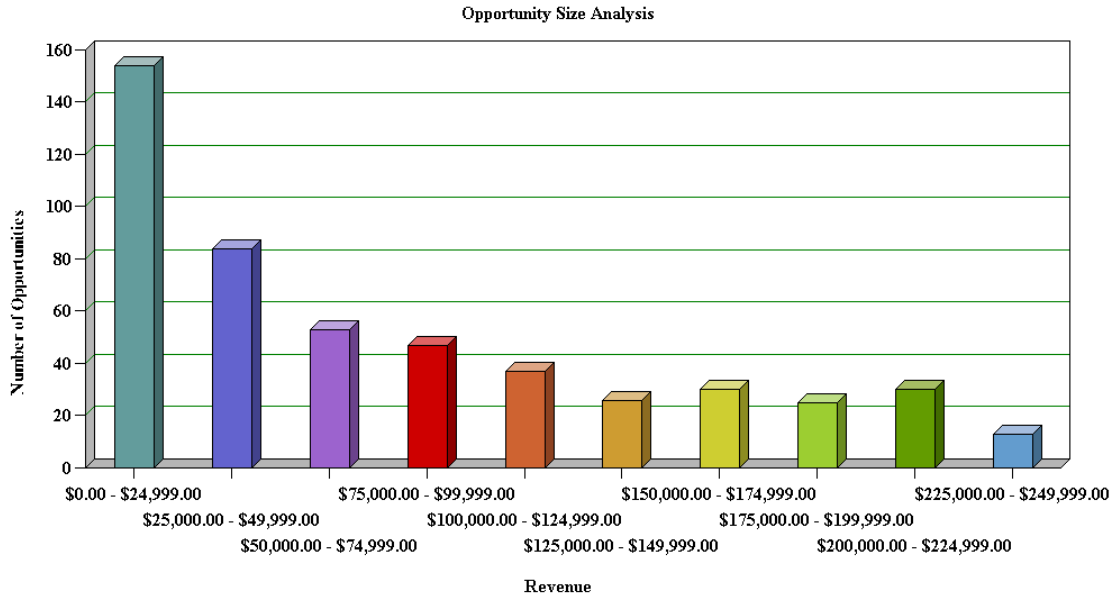


Figure 193. 3dBar Chart

If the chart is configured with a Z (series) axis, a cluster of bars appears for categories rather than a single bar. This is shown in [Figure 194](#).

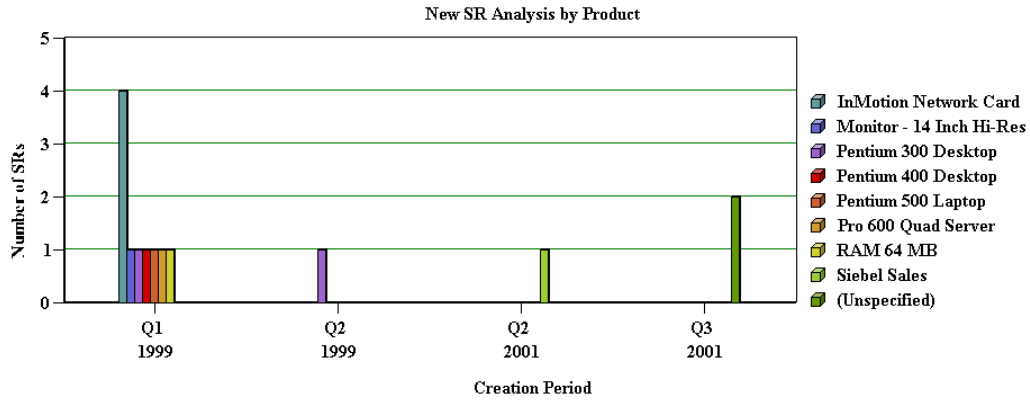


Figure 194. 3dBar Chart with Series Axis

- **3dHorizBar.** A 3dHorizBar chart is functionally equivalent to a 3dBar chart, but has the X and Y axes switched, with the result that the bars are horizontal. A 3dHorizBar chart appears in [Figure 195](#).

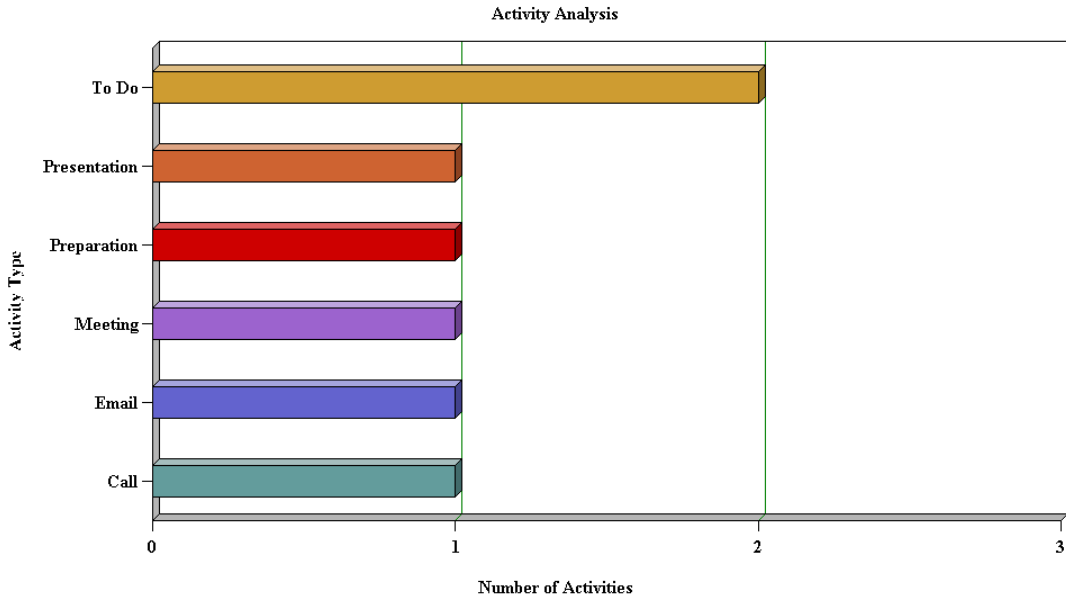


Figure 195. 3dHorizBar Chart

The individual horizontal bars are replaced by clusters of horizontal bars if a series axis is present, as shown in [Figure 196](#).

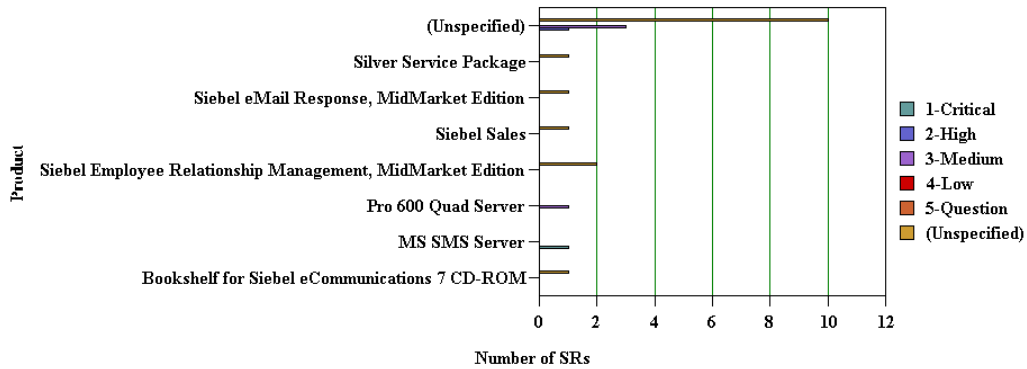


Figure 196. 3dHorizBar Chart with Series Axis

- **3dStackedBar.** A 3dStackedBar chart normally has a series axis. The chart displays a single stack of bars for each category, within which appears a bar of a different color for each series. Stacked bar charts are useful for seeing the individual value for each series within the category as well as their total for the category. An example of a 3dStackedBar chart appears in [Figure 197](#).

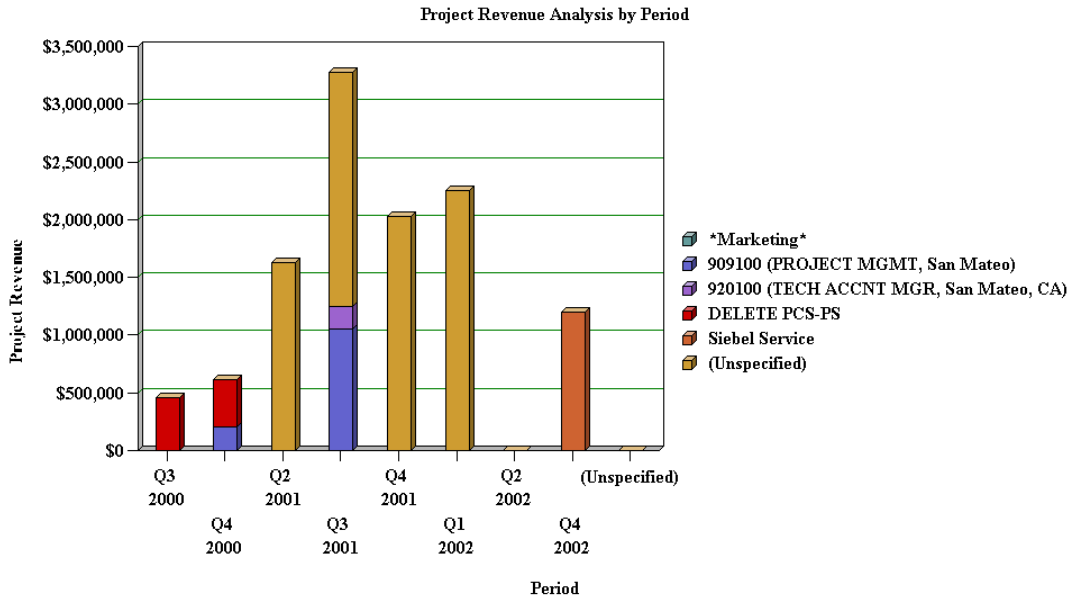


Figure 197. 3dStackedBar Chart

This figure displays a Project Revenue Analysis chart. The data values axis corresponds to project revenue, the category axis corresponds to a quarter, and the series axis corresponds to the project name. So for each quarter along the X axis, there is a stack of bars. Each bar in the stack indicates the revenue reached in a particular quarter. The stacks within each bar indicate the individual projects.

- 2dBar.** A 2dBar chart is functionally equivalent to a 3dBar chart, but is displayed without the illusion of depth. Two-dimensional charts are generally easier to read accurately, but may seem less visually attractive than their three-dimensional counterparts. A 2dBar chart appears in [Figure 198](#).

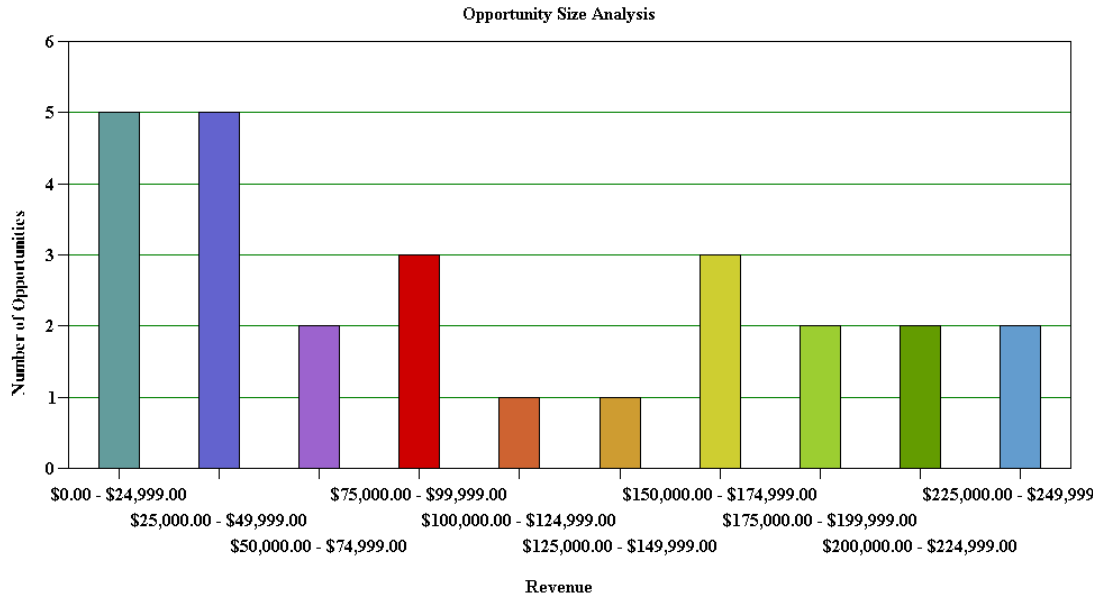


Figure 198. 2dBar Chart

Like the 3dBar chart, a 2dBar chart displays bars in clusters if a series axis is present.

- **2dHorizBar.** The 2dHorizBar chart type is functionally equivalent to the 3dHorizBar type, but is displayed without the illusion of depth. A sample 2dHorizBar chart appears in [Figure 199](#).

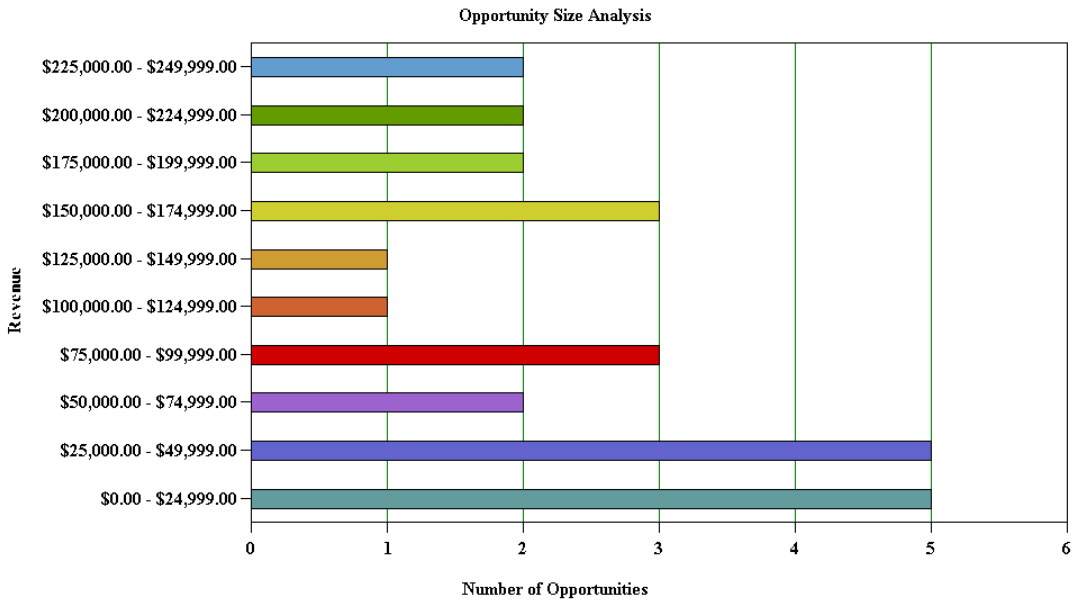


Figure 199. 2dHorizBar Chart

- **2dStackedBar.** The 2dStackedBar chart type is functionally equivalent to the 3dStackedBar type, but is displayed without the illusion of depth. A sample 2dStackedBar chart appears in [Figure 200](#).

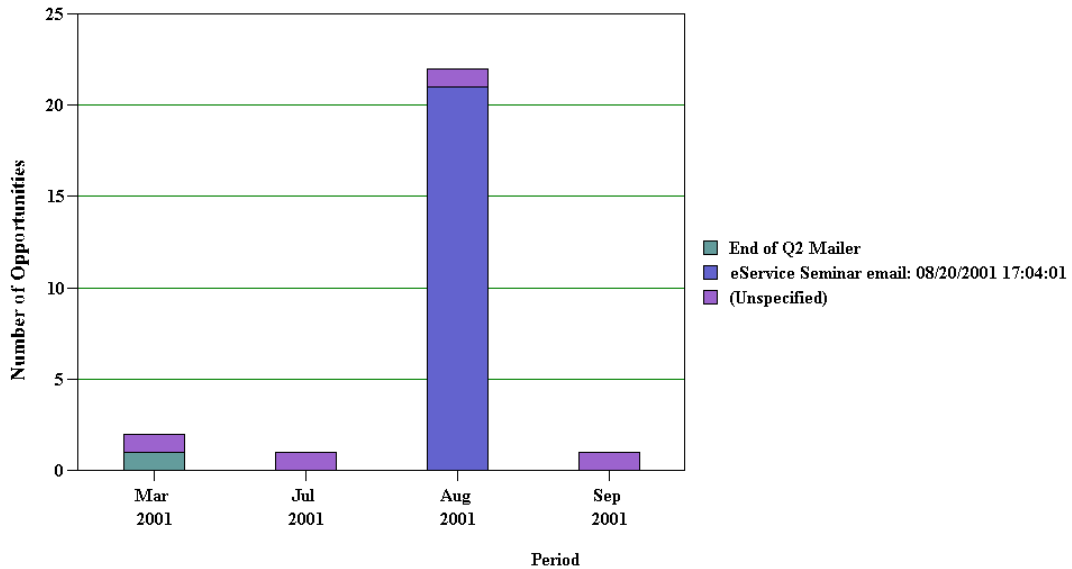


Figure 200. 2dStackedBar Chart

Line Charts

Line Charts are used to observe trends across categories or over time.

- **2dLine.** The 2dLine chart type displays one or more line curves plotted against the X-Y grid. If there is no series axis, a single line curve appears. If there is a series axis, one line curve appears for each color in the legend. A 2dLine chart appears in [Figure 201](#).

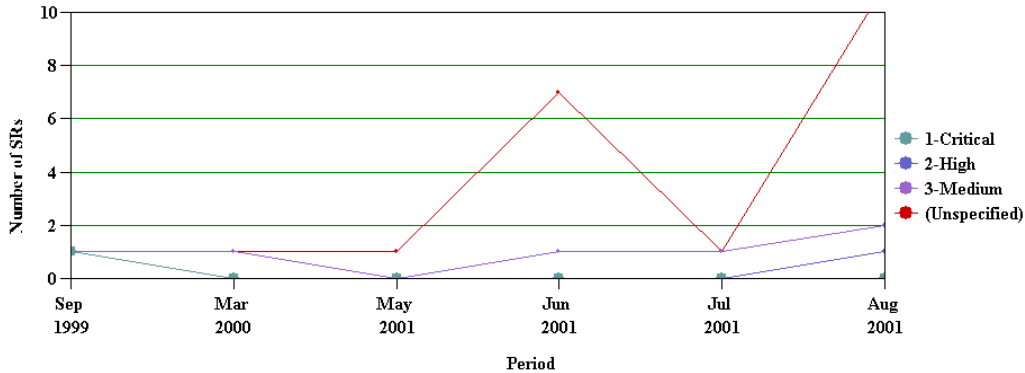


Figure 201. 2dLine Chart

- **3dLine.** The 3dLine chart type is functionally equivalent to the 2dLine type, but appears with the illusion of depth. A 3dLine chart (showing the same data as the 2dLine chart in [Figure 201](#)) appears in [Figure 202](#).

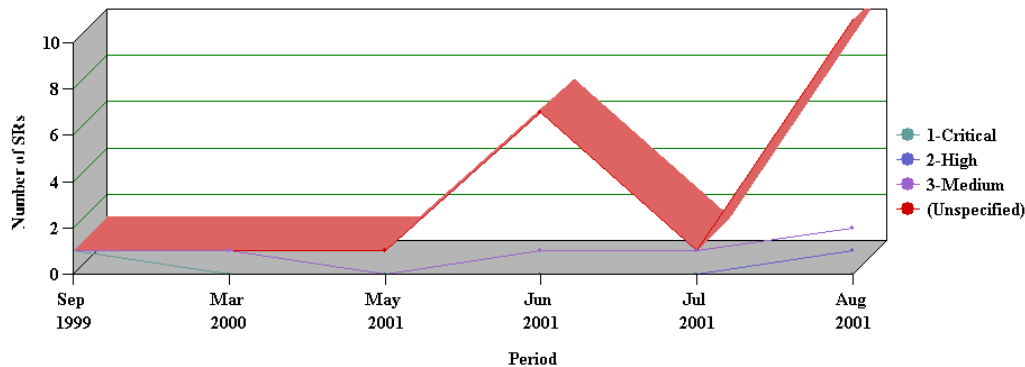


Figure 202. 3dLine Chart

- **2dSpline.** The 2dSpline chart type displays one or more line curves plotted against the X-Y grid, with the points plotted accurately but the line between them smoothed mathematically. If there is no series axis, a single curve and set of points appear. If there is a series axis, one curve and corresponding set of points appear for each color in the legend. A 2dSpline chart appears in [Figure 203](#).

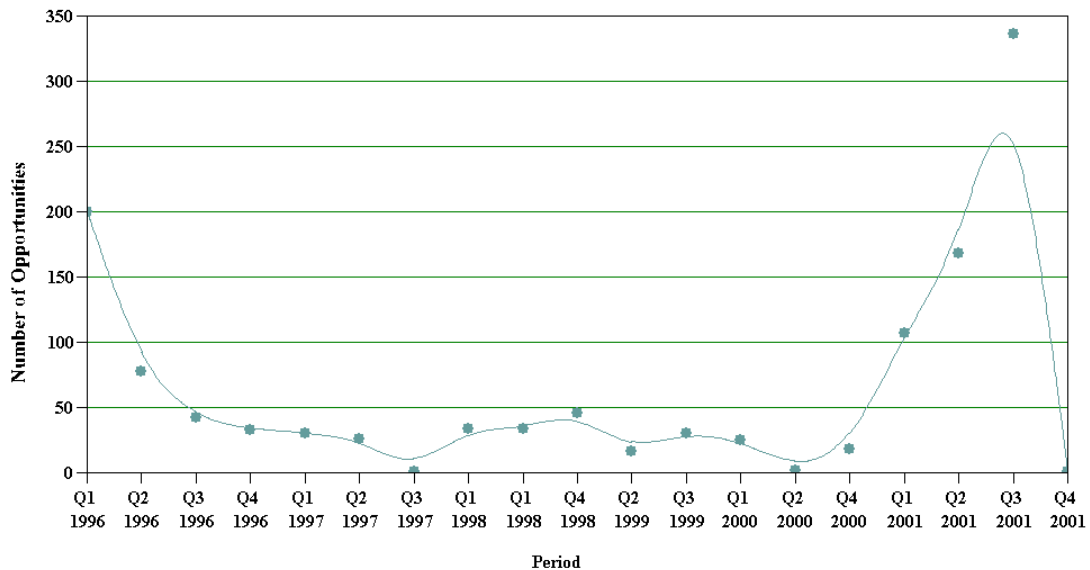


Figure 203. 2dSpline Chart

- **3dSpline.** The 3dSpline chart type is functionally equivalent to the 2dSpline type, but appears with the illusion of depth, and does not display the actual data points, only the smoothed curve. A 3dSpline chart (showing the same data as the 2dSpline chart in [Figure 203](#)) appears in [Figure 204](#).

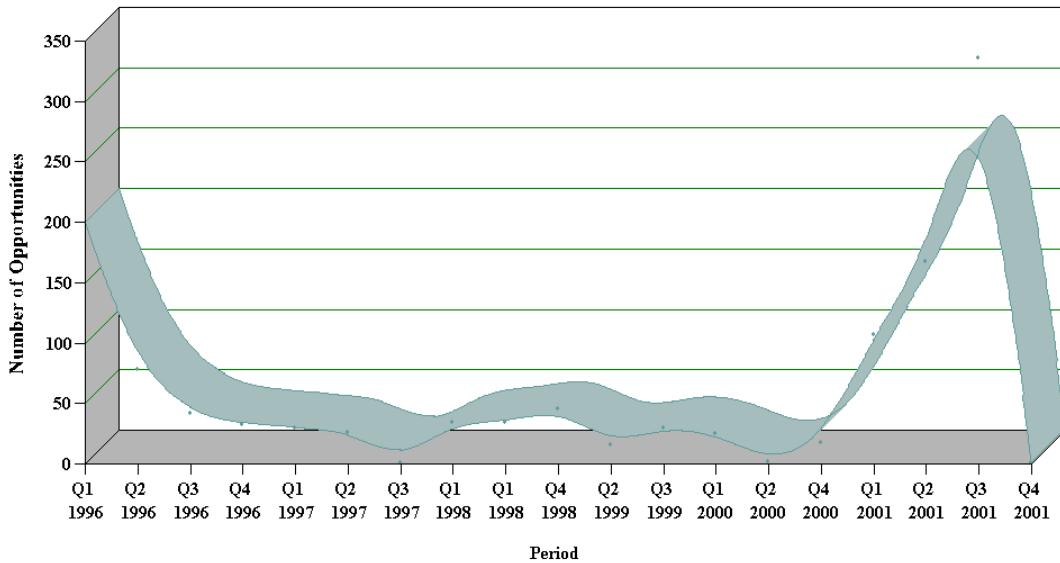


Figure 204. 3dSpline Chart

- Combo.** A chart of the Combo type displays a single bar chart with dots superimposed on it. The two charts share the category axis, but each has its own data points axis (on the left for the bar chart, and on the right for the line chart). A sample Combo chart appears in [Figure 205](#).

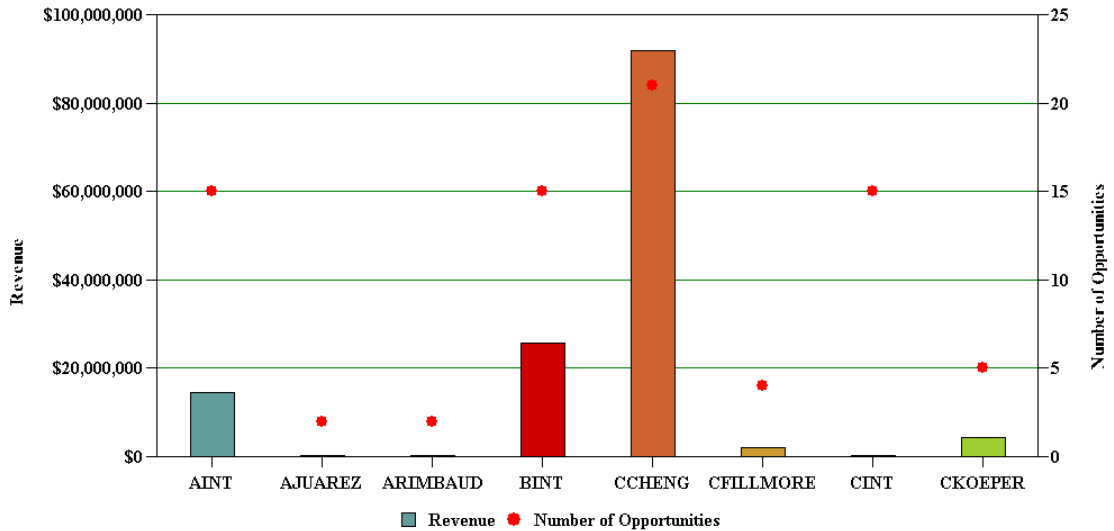


Figure 205. Combo Chart

Pie Charts

Pie Charts are used to compare the relative difference across categories by dividing a circle into segments that represent each category's percentage of the whole.

- **3dPie.** The 3dPie chart type aggregates data point data in the records by category, and displays each category as a separate segment in the pie. The category (X) axis is the set of pie slices and corresponding labels. The data points (Y) axis determines the relative size of each pie slice as a percentage of the total. You cannot specify a series axis for pie charts. The 3dPie chart type gives the illusion of depth, for visual attractiveness. A sample 3dPie chart appears in [Figure 206](#).

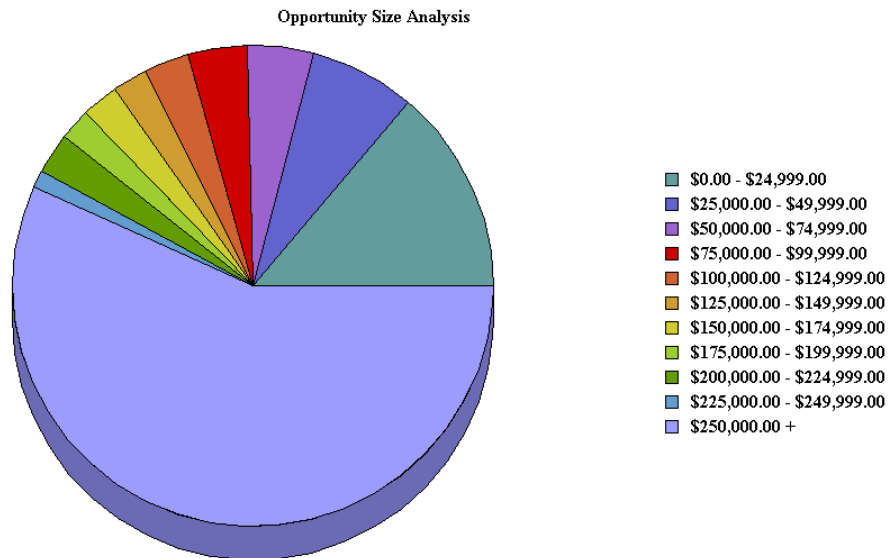


Figure 206. 3dPie Chart

- **2dPie.** The 2dPie chart type is functionally the same as the 3dPie type, but without the illusion of depth. A sample 2dPie chart appears in [Figure 207](#).

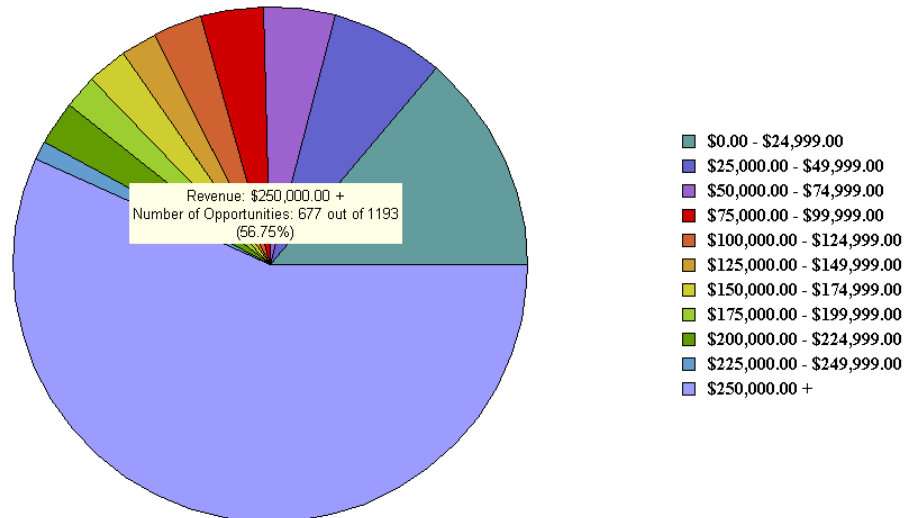


Figure 207. 2dPie Chart

Scatter Charts

- **2dScatter.** A scatter chart—a chart with the 2dScatter type—displays the distribution of data according to two attributes. This is useful for probability distributions, among other applications. The category axis must contain numeric, as opposed to date or text data. This makes the 2dScatter type unsuitable for conversion to other chart types such as bar, line, or pie. For this reason, the 2dScatter type does not appear in Type picklists, and a 2dScatter chart does not have a Type picklist. A 2dScatter chart appears in [Figure 208](#).

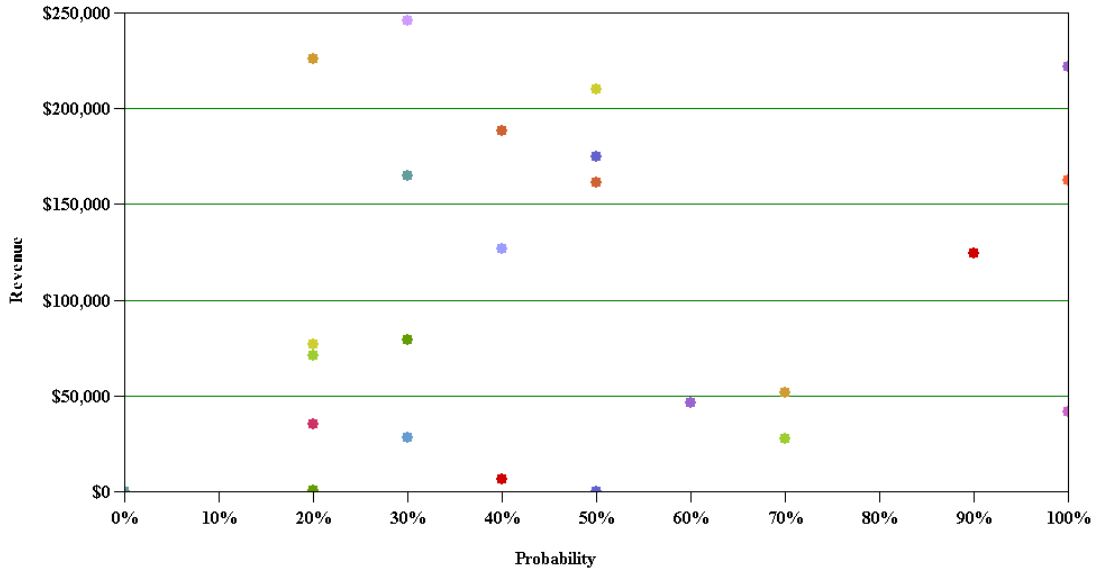


Figure 208. 2dScatter Chart

Configuring Chart Applets

A chart is built as an applet containing one or more Chart object definitions. The Chart object type is a child of applet. The Chart object type has Chart Element children. This section describes how chart applets are configured.

You can also use the Chart Applet Wizard to create chart applets. See [“Using the Chart Applet Wizard” on page 751](#).

Business Component Mapping

A chart applet has, like all applets, a business component identified in its Business Component property. Records in this business component—subject to the current view, the current query, and visibility considerations—provide the data displayed in the applet. In the case of a chart applet, specific fields are used to provide the data for the category, data point, and series axes. The correspondence between axes and fields is specified in properties in the Chart object definition.

In the simplest case—a single bar or line graph, with no series axis—a category field and a data point field are specified. Pairs of category and data point field values are plotted as points or bars. If multiple records have the same category value, their data point values are added together.

Special-Purpose Applets and Controls

Chart Applets

The Oppty Chart Applet - Source Analysis applet provides an illustration of this process ([Figure 209](#)).

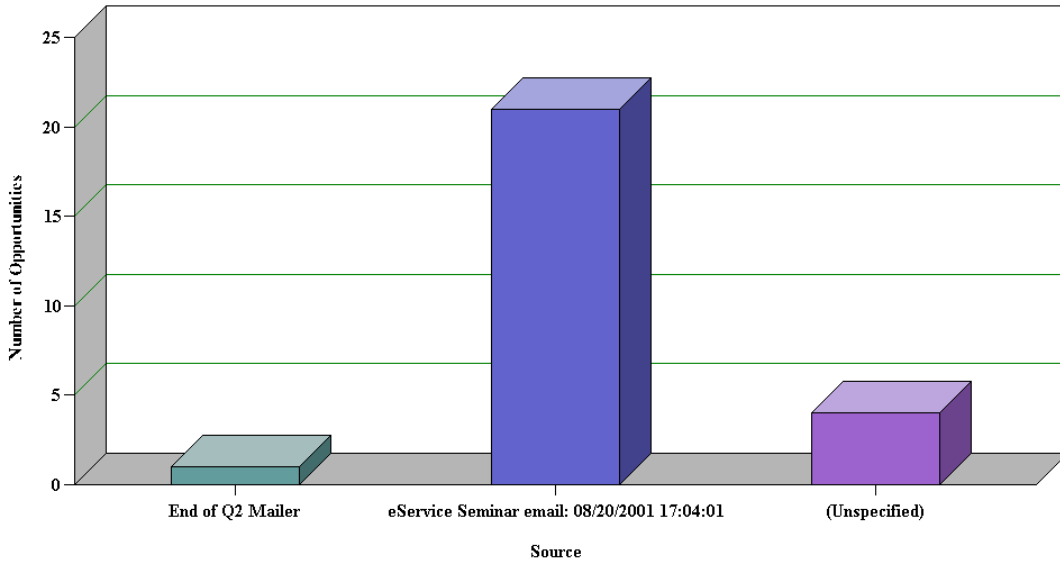


Figure 209. Oppty Chart Applet - Source Analysis

This applet displays the number of opportunities on the data point axis plotted against the source of the opportunity (referral, magazine article, Web site, and so on) on the category axis. To generate the data required for the curve, the Source field in each record is checked and the number of opportunities for each distinct source value is tallied. The result is a two-row temporary table with a column for each source, as shown in Figure 210.

		category (Source)		
heading count		Mailer	Seminar eMail	Unspecified
		1	21	4

data point value

Figure 210. Temporary Table for Single-Curve Chart Data

For a multiple-curve chart, a row is added to the temporary table for each curve in the series (Figure 211).

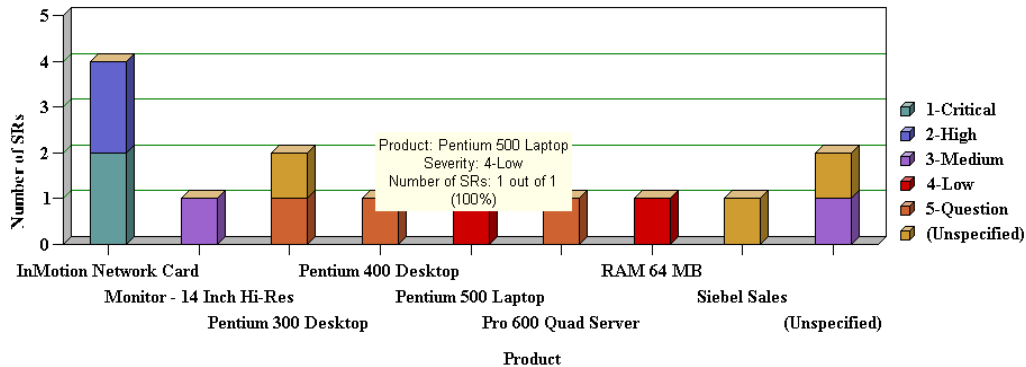


Figure 211. Multiple-Curve Chart

The temporary table for a multiple-curve chart is illustrated in [Figure 212](#).

	category (Product)				
	Network Card	Monitor	Pentium 300	Pentium 400	Pentium 500
Critical	2	0	0	0	0
High	2	0	0	0	0
Medium	0	1	0	0	0
Low	0	0	0	0	0
Question	0	0	1	1	1
Un-specified	0	0	1	0	0

Figure 212. Temporary Table for Multiple-Curve Chart Data

To define the data mapping from the business component into the chart applet, you need to define the following properties in the Chart object:

- **Category Field.** Contains the name of a text or date field in the business component (except for scatter charts, which use a numeric category field). When the business component records are scanned, the different values found in this field are mapped into different categories. These values are displayed on the chart's X-axis labels.
- **Data Point Field.** Contains the name of a numeric field in the business component, or is unspecified. If specified, the value in this field in each record is added to the total for the category field value in the same record. If a data point field is not specified, the count for the corresponding category field is incremented rather than adding the data point value to the total for the category field. These counts or totals determine the height along the Y-axis of a bar or line curve point for each unique category field value in the curve. Rather than a total or a count, some other function (specified in the Data Function property) may determine the use of the data point field data.

- **Series Field.** Contains the name of a text field in the business component, or is unspecified. When the business component records are scanned, the different values found in this field are mapped into different curves. These values are displayed on the chart's legend labels.

NOTE: The maximum number of Series cannot exceed 50 when running the chart. If it does exceed 50, an error message is displayed. The user may have to run another query that results in less than 50 Series.

- **Data Function.** The Data Function property determines how the data point field values are converted into the new table's cell values. Possible values are Sum (simple addition), Count (number of occurrences of a cell value), Average (average value per record), and Plot (different from Count only in that when a cell is empty, it is charted as NULL instead of 0).

The preceding descriptions cover the use of these properties for the most general cases. There are a number of special cases in which these properties are configured differently than described. Some special case configuration scenarios are described in the sections that follow. For descriptions of the properties, see *Object Types Reference*.

Configuring the Picklists

A chart applet typically provides one or more picklists along the upper edge that allow the user to reconfigure the chart's presentation or use of data. These picklists are illustrated in [Figure 213](#).

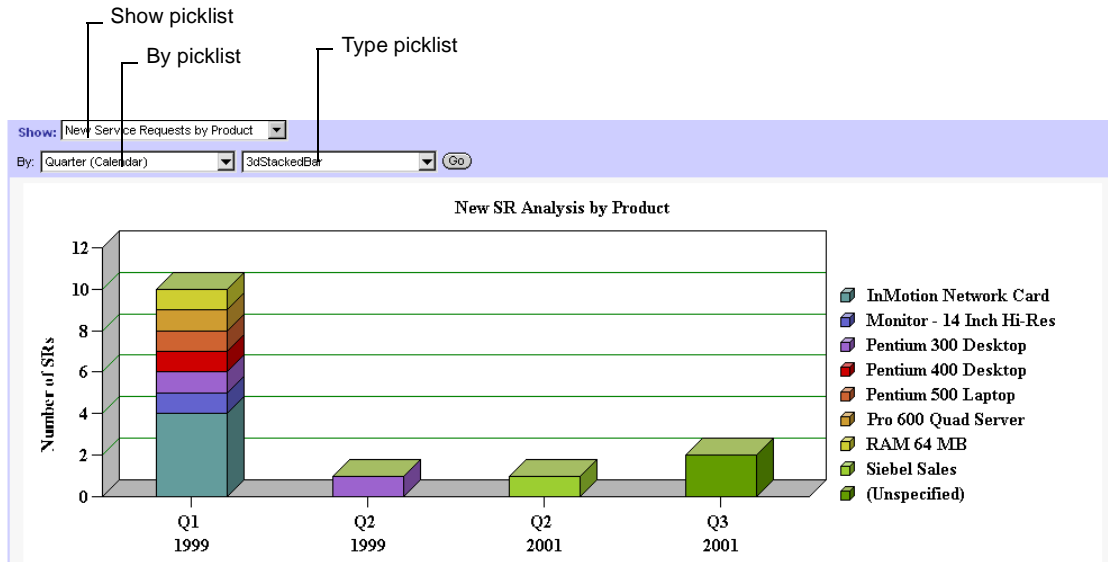


Figure 213. Picklists in a Chart Applet

These picklists are described as follows:

- **Type picklists.** This is the most common of the four picklists, and appears in most chart applets. It provides the user with the means to select a different type of chart for the same data, such as a pie chart instead of a bar chart, or a two-dimensional line chart instead of a three-dimensional one. The chart types are described in detail in [“Chart Layout Options” on page 722](#).

The options for the Type picklist are specified in the Picklist Types property of the Chart object definition, as a comma-separated list of chart type names such as the following:

`3dBar, 3dStackedBar, 3dPie, 3dHorizBar, 2dBar, 2dStackedBar, 2dPie, 2dHorizBar`

There cannot be any spaces between the elements in the comma-separated list.

The default type—the chart type to appear when the chart is initially displayed—is specified in the Type property. Charts without a Type picklist use the Type property to specify the chart type of the chart; in that situation the chart type cannot be changed by the user.

- **Show picklists.** This picklist allows the user to change what is displayed on the Y axis. The choices available depend on the configuration of certain properties in the Chart object definition. The Show picklist displays a selection list of field/function combinations which determines what values are plotted along the Y axis.

For information on configuring the Show picklist, refer to [“Configuring Show Picklists” on page 744](#).

- **By Picklist.** This picklist allows the user to change what is displayed on the X axis. This can provide any one of three roles, depending on the configuration of certain properties in the Chart object definition:
 - In period Charts, the By picklist is populated with different periods. This allows the user to select from a list of possible X-axis periods for calendar (day/week/month/quarter/year) data. This requires selection options to be specified in the Picklist Periods property in the Chart object definition.
 - When a list of source fields is specified rather than a single source field, the picklist allows the user to choose which source field populates the X axis.
 - It can allow the user to invert the X and Z axes, so the user can see the data from a source field in a business component displayed along the X or Z axis per the picklist selection.

For information on configuring the By picklist, refer to [“Configuring the By Picklist” on page 746](#).

- **Second “By” picklists.** This picklist allows the user to choose which source field populates the Z axis. For information on configuring the second By picklist, refer to [“Configuring the Second By Picklist” on page 747](#).

Each of the four picklists requires a corresponding control of type ComboBox, as a child object definition of the chart applet. Each has required values in the Name and MethodInvoked properties, as detailed in [Table 48](#).

Table 48. Name and MethodInvoked Properties for Four ComboBox Controls

Picklist	Control Name	MethodInvoked
Type	ChartPicktype	PickChartType
Show	ChartPickfunction	PickYAxis
By	ChartPickby	PickXAxis
By #2	ChartPickby2	PickZAxis

Configuring Show Picklists

The Show picklist (the combo box control named ChartPickfunction) can be configured to display a selection list of field/function combinations, the selection from which determines what values are plotted along the Y axis. Multiple combinations of source field and function are provided in the selection list. The Y axis title is obtained from the text in the user's Show picklist selection.

To configure the Show picklist, the following three properties of the Chart object definition are used:

- **Data Point Field.** You enter a comma-separated list of source fields, one for each entry that is to appear in the Show picklist. The first entry in the list is the default. If only one field name is entered, it applies to all functions in the picklist.
- **Data Function.** You enter a comma-separated list consisting of the following function names: SUM, COUNT, AVERAGE, or PLOT. PLOT indicates that the Y values are derived directly from the values in the source field. The order in the comma list determines the association with a data point field and title (picklist function). If the comma-separated list is omitted or it contains fewer elements than the list of names in the Picklist Functions property, the list `Sum,Count,Average,Plot` is substituted.
- **Picklist Functions.** You enter a comma-separated list of Y-axis titles, which are also the text which appears in the picklist. The order in the comma list determines the association with a data point field and data function.

For example, you could configure a Show picklist with explicit syntax that offers three choices: Number of Opportunities, Opportunity Revenue and Opportunity Expected Revenue. This is configured with the property settings shown in [Table 49](#).

Table 49. Show Picklist Properties for Sales Method Bar Chart

Property	Value
Picklist Functions	Number of Opportunities, Opportunity Revenue, Opportunity Expected Revenue
Data Function	Count,Sum,Sum
Data Point Field	Name,Revenue,Expected Revenue

As can be seen from the table, there are three values in each comma-separated list. The first entry, Number of Opportunities, performs a Count function on the Name field. The second entry, Opportunity Revenue, performs a Sum function on the Revenue field. The third entry, Opportunity Expected Revenue, performs a Sum function on the Expected Revenue field.

An example of a Show picklist configured with implicit syntax and the standard function list is in the Lead Source Analysis chart in the Opportunity New Business Analysis view in Siebel Sales (Oppty Chart Applet - New Business). The picklist offers three choices: Number of Opportunities, Opportunity Revenue, and Average Opportunity Revenue. This is configured with the property settings shown in [Table 50](#).

Table 50. Show Picklist Properties for Lead Source Analysis Chart

Property	Value
Picklist Functions	Number of Opportunities,Opportunity Revenue, Avg Opportunity Revenue
Data Function	Count
Data Point Field	Revenue

The value of Revenue in the Data Point Field property applies to all entries in the picklist.

The value of Count in the Data Function property is unnecessary; it could be left blank instead. Whenever the number of entries in the Data Function property is not the same as the number in the Picklist Functions property, the system supplies a standard Data Function list. This list is the following:

Count , Sum , Average , Plot

The first picklist entry, Number of Opportunities, performs a Count function on the Revenue field. The second entry, Opportunity Revenue, performs a Sum function on the same field. The third entry, Avg Opportunity Revenue, performs an Average function.

This means of configuring Show picklist behavior predates the ability to specify triplets of name, function, and field, and is more restrictive. It has been retained for backwards compatibility with earlier versions of Siebel applications. Generally it makes more sense to explicitly specify the values in the three properties.

Configuring the By Picklist

The contents of the Category Field property in the Chart object definition determine the behavior of the By picklist (ChartPickBy combo box control), as follows:

- **Calendar increments in the picklist and X axis.** If the Category Field property contains the name of a single field that has a DTYPE_DATE data type, the X axis displays calendar increments and the chart is considered a *period chart*. In this situation, the picklist is populated with calendar increment options, including user defined periods (specified in View > Site Map > Application Administration > Periods) such as Day, Week, Month, Quarter, and Year.

For example, in the New Business Analysis chart, the category field is Created (the date of creation of the record, hence of the opportunity). As a result, the category axis contains date increments, based on the increment the user selects in the By picklist.

- **Text labels in the X axis, category and series field names in the picklist.** If the Category Field property contains the name of a single text field from the business component, and a series field has also been specified (in the Series Field property), the By picklist is populated with the names of the category field and the series field. The user can select either field to populate the X axis with labels derived from the contents of that field; the unselected field populates the legend box (Z axis) with labels. The category field is the default, and is initially displayed on the X axis.

For example, the chart in the Service Request Product Analysis view in Siebel Service has a category field of Product and a series field of Severity. When the chart is initially displayed, the X axis labels are product names and the legend labels are severity levels. However, the field names Product and Severity appear in the By picklist, and the latter selection allows the user to display severity levels in the X axis and product names in the legend.

- **Text Labels in the X axis, multiple field names in the picklist.** If the Category Field property contains a comma-separated list of field names, the user is provided with this list of fields at run time in the By picklist. The user's selection determines the field which populates the X axis. The first value in the comma-separated list is the default. (You should avoid blank spaces before or after field names in the list.)
- **Numeric values in the X axis, no picklist.** If the Category Field property contains the name of a single numeric field, the X axis is populated with numeric increments, similar to the process of generating increments for the Y axis. In this situation, the By picklist is not shown.

For example, the Probability Cluster Analysis chart in the Opportunity Probability Cluster Analysis view has a category field of Rep % (the probability of a sale). In this chart, probability is plotted against the X axis, the X axis increments are percentages from 0% to 100%, and no By picklist appears.

Configuring the Second By Picklist

The contents of the Series Field property in the Chart object definition determine the behavior of the second By picklist (the combo box control named ChartPickBy2), as follows:

- If the Series Field property is blank, all records are mapped into a single series.

- If the Series Field property contains the name of a field from a business component, the Z axis (legend) is populated with labels derived from the contents of that field.
- If the Series Field property contains a comma-separated list of field names, the user is provided with this list of fields at run time in the second By picklist. The user's selection determines the field which populates the Z axis. The first value in the comma-separated list is the default.

Charts with Multiple Curves Plotted Against One Y Axis

Multiple line graph curves can be plotted against the same Y axis, based on different source field/function combinations. The name for each curve appears in the legend. For example, you may want revenue, expected revenue, and net profit to appear as superimposed curves on the same line graph. To accomplish this, set the following property values in the Chart object definition:

- **Data Point Field.** Provide a comma-separated list of source fields, one for each curve to appear in the graph.
- **Data Function.** Provide a comma-separated list consisting of some of the following function names: SUM, COUNT, AVERAGE, or PLOT. PLOT indicates that the Y values are derived directly from the values in the source field. The list of function names must have the same number of entries as the Data Point Field list. The order in the comma list determines the association with a data point field and title.
- **Picklist Functions.** Provide a comma-separated list of Y-axis titles, which identify the individual curves in the Legend. The list of titles must have the same number of entries as the Data Point Field list. The order in the comma list determines the association with a data point field and data function.
- **Series Field.** Remove any existing value(s) from this property; it must be blank. Otherwise, the multiple curves are converted to a Z axis.
- **Multi Data Point.** Set to TRUE. This indicates that multiple curves are to be plotted.

You should also remove the Show combo box and its label in the Applet Web Editor.

Charts with Two Y Axes

Two line graph curves can appear in the same Chart, plotted against different Y axes (one to the left of the graph, the other to the right). Any field/function combination can be used on the left Y axis, and likewise for the right. To accomplish this, set the following property values in the Chart object definition:

- **Data Point Field.** Specify two fields, separated by a comma. The first is for the left Y axis, the second is for the right Y axis.
- **Data Function.** Specify two functions, separated by a comma. The first is for the left Y axis, the second is for the right Y axis.
- **Type.** Set to Combo.

Axis Points—Limiting and Sorting

The number of X axis (category) or Z axis (series) labels can be limited to some predefined number. This can be useful if you are interested in displaying only the *N* highest or *N* lowest values for some field or calculated Y value. For example, you could display the 10 highest revenue accounts by charting the Revenue field in descending order and limiting the X axis to 10 data points. This is accomplished using two properties of the axis label Chart Element for the appropriate axes, as follows:

- **Divisions property (X or Z axis).** Enter an integer to limit the number of X axis or Z axis labels to the number you enter. Note that the AxisId property must be either XAxis or ZAxis, and the Type property must be AxisLabel.
- **SortSpecification property (Y axis).** Enter a value of Ascending or Descending. Note that the AxisId property must be set to YAxis and the Type property must be AxisLabel.

You can set up a sort specification on the Y axis independent of limiting the number of X or Z axis divisions. A sort specification on Y will order the data points regardless of whether you are limiting the display to the first *N* points. The converse is not true, however; it would not make sense to set a number of X or Z axis divisions without also setting a sort specification on Y.

You also can sort on X axis or Z axis labels instead of Y axis values. To accomplish this, you set the Sort Specification in the X axis (or Z axis) label Chart Element object definition rather than in the Y axis label. For example, if the X axis is displaying country names, they can appear alphabetically from left to right. This is different from sorting on Y axis values, which are numeric values from a field in a business component or function based on that field.

Chart Element Object Type

Chart Element is a child object type of Chart. The following types of Chart Elements (as specified in the Type field in the Chart Element object type) are supported:

- **AxisLabel.** Displayed along each axis, with one label for each division of the axis.
- **AxisLineGrid.** Grids make it easier to comprehend a Chart. You can set various grid properties, such as grid color, width, and visibility, on an axis-by-axis basis.
- **AxisTitle.** Displayed along each axis, with one title per axis.
- **Graphic.** A line, rectangle, or ellipse used to emphasize a region of the Chart.
- **Legend.** The list of colored rectangles with accompanying category labels on the left side of the Chart.
- **Plot.** The area that contains the graphs, usually in the center of the Chart.
- **Title.** The large string of text, usually at the top of a Chart.
- **Font, Color, and Size.** For most Chart Elements that contain text, you can set such text properties as font, color, and size.
- **Fill color.** You can set the fill color of the Chart and Plot Chart Element types.

The properties of the Chart Element that apply to the axis label for the X axis (Coordinates, Display Format, Divisions, List Of Values, Sort Specification, and Text) should not be used when specifying a list of X axis source fields, as they can be relevant only for one X axis field. Also, the text of the X axis title is determined dynamically from the combo box selection if the By combo box provides a list of source fields. Whatever is in the Text property in the AxisTitle chart element for the X axis is overridden at run time.

The same restrictions are relevant for the Z axis.

Making X-Axis Labels Vertical

You can make x-axis labels vertical so that they do not overlap with each other. To do this, set the Vertical property to TRUE for the Chart Element object whose Axis Id property is set to XAxis.

Sizing Chart Images

You can change the size of a chart applet by setting the HTML Width and HTML Height properties (in pixels) for the Chart control child object of the applet.

The default values are 1012 for HTML Width and 560 for HTML Height.

Performance Considerations

When a chart is traversing records in the business component, its progress is indicated at the bottom of the window. Since traversing all of the records of a business component can be time-consuming, charts are not well suited for data sets larger than 1,000 records.

Various factors affect the performance of charts in Siebel applications:

- The number of records in the business component
- Whether the chart needs to search a multi-value group to obtain its data
- Whether a data point field is specified
- If the data point field is a currency field, the number of records whose currency is not the functional currency
- The processor, operating system, and database system

Using the Chart Applet Wizard

The following procedure identifies the steps required to create a new chart applet using the Chart Applet Wizard.

To create a new chart applet

- 1 Choose File > New Object.

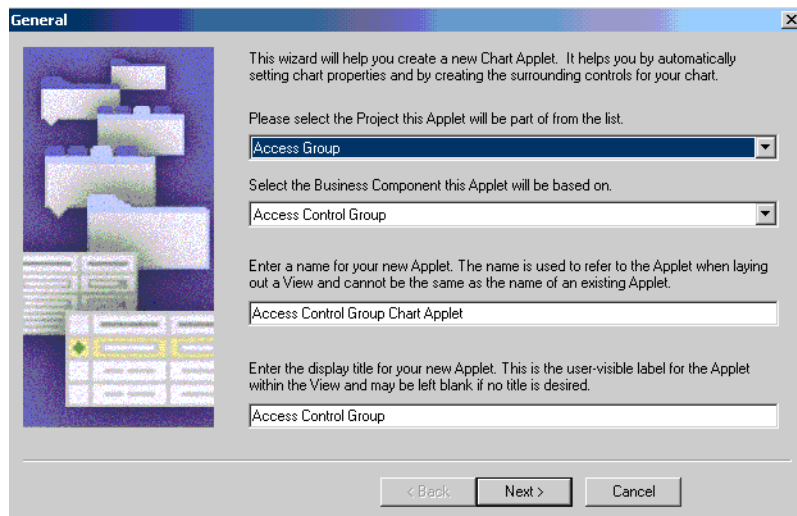
The New Object Wizard dialog box appears with the list of objects that can be created through wizards.

- 2 Select the Applets tab and then select the Chart Applet icon.

The Chart Applet Wizard appears.

- 3 In the General dialog box, complete the following information and then click Next:

- Project
- Business Component
- Name
- Display Name



General

This wizard will help you create a new Chart Applet. It helps you by automatically setting chart properties and by creating the surrounding controls for your chart.

Please select the Project this Applet will be part of from the list.

Access Group

Select the Business Component this Applet will be based on.

Access Control Group

Enter a name for your new Applet. The name is used to refer to the Applet when laying out a View and cannot be the same as the name of an existing Applet.

Access Control Group Chart Applet

Enter the display title for your new Applet. This is the user-visible label for the Applet within the View and may be left blank if no title is desired.

Access Control Group

< Back Next > Cancel

- 4 In the Y Axis dialog box, enter the options that will be displayed on the Y axis and then click Next.

For more information about the Y Axis, see [“Axis Terminology”](#) on page 720.

The options below describe what will be shown on the Y axis (the vertical axis).
By default, the Y axis will display the number of records (in the current business component query) for each X axis category. For example, "Number of Opportunities!"
If you want the Y axis to display the summation or average of values in a field for each X axis mark, select the desired field in the Data point field picklist, and select Sum or Average in the Function picklist

Data point field: Function:

If you specify more than one title, the end user will be able to select which one he wants to see in the "Show:" combo box.

Titles

Count:

Sum:

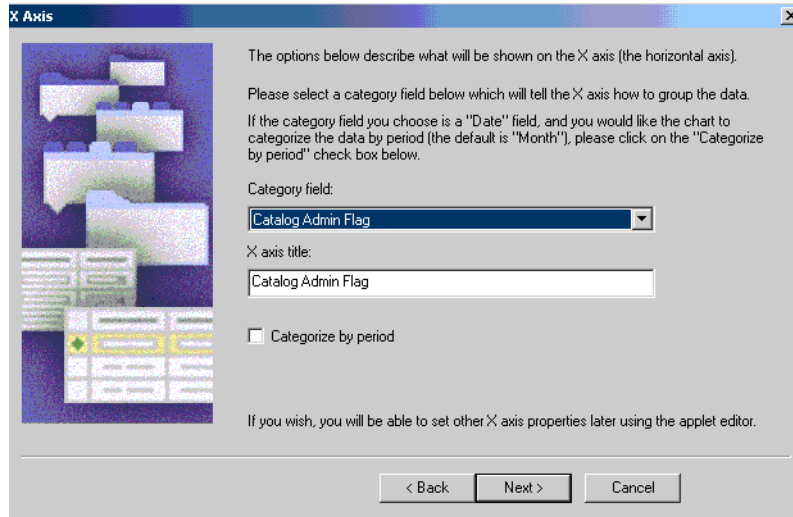
Average:

If you wish, you will be able to set other Y axis properties later using the Applet editor.

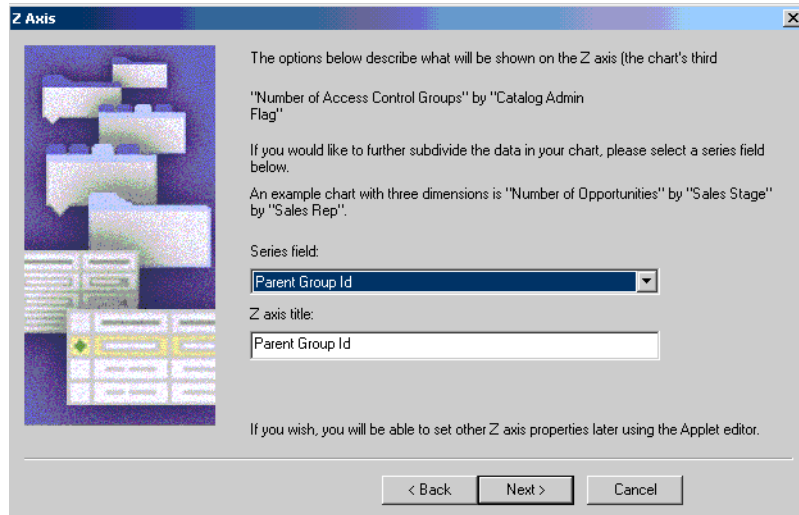
< Back Next > Cancel

- 5 In the X Axis dialog box, enter the necessary information for the X Axis and then click Next.

For more information about the X Axis, see [“Axis Terminology”](#) on page 720.



- 6 In the Z Axis dialog box, enter the necessary information, and then Click Next.
For more information about the Z Axis see [“Axis Terminology” on page 720](#).



- 7 In the Chart Title dialog box, enter a title, and then click Next.
- 8 In the Web-Layout-General dialog box, select the template that will be associated for the base read-only mode and then click Next.
- 9 In the Finish dialog box review the information and then click Finish.

The Chart Applet Wizard creates the necessary object definitions and sets the property values based on information you entered in the wizard. The Web Applet Layout Editor opens allowing you to map controls to placeholders in the Web template.

For more information, see [“Editing the Layout of Web Page Objects” on page 579](#).

Tree Applets

A tree applet is used to create a view, called an explorer view, that allows the user to navigate hierarchically through a structured list of records of related business components. An example of a tree applet and explorer view in Siebel Service is the Service Requests applet in the Service Request Explorer view, shown in [Figure 214](#).

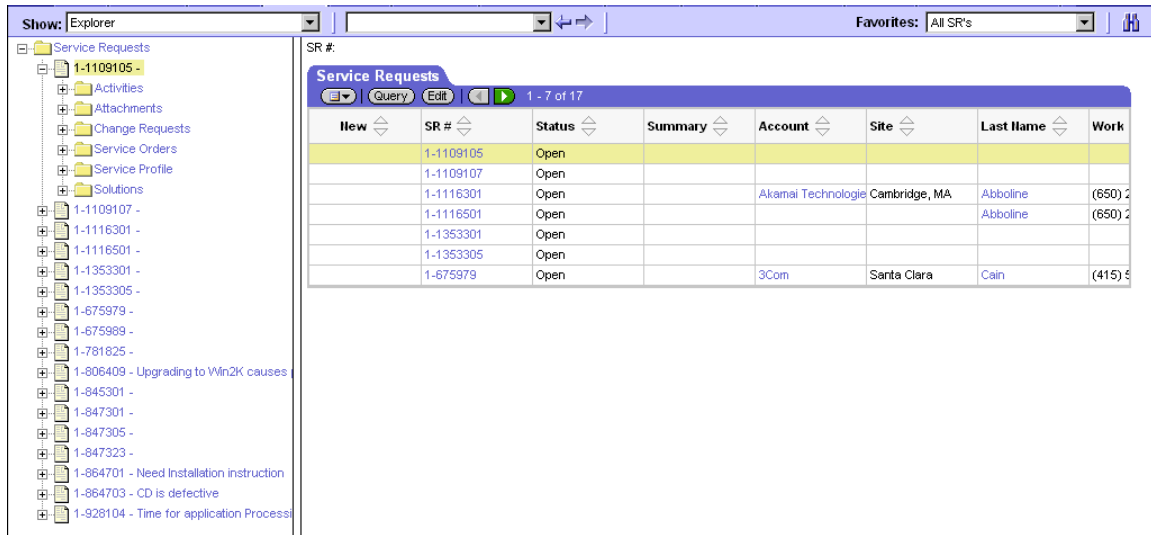


Figure 214. Service Requests Explorer View

This view (SR Explorer View) contains a tree applet (SR Tree Applet) in the left side, and one of various predefined list applets in the right side. The particular list applet that appears on the right depends on which node is selected in the tree on the left. For example, if the user double-clicks on the Change Requests folder in the tree hierarchy, the list applet on the right changes to display change requests records.

A tree applet in an explorer view is similar in operation to the Object Explorer and Object List Editor in Siebel Tools. The user may expand and collapse folders in the tree applet, and view the records in that folder in the list applet. The hierarchy displayed in the tree applet represents master-detail relationships between records of different business components.

For example, when the user expands a service request (document icon) by double-clicking, a set of folders appears hierarchically beneath it including Activities, Attachments, Change Requests, Solutions and so on. When the user expands one of these child folders, a list of records appears of the corresponding business component. If the user expands the folder for a service request, and then expands the Activities folder beneath it, the list of records displayed is the set of Activities for that service request. In the master-detail relationship between service requests and Activities, these Activity records are detail records of the master service request record that was expanded.

The user can also add or associate detail records of various kinds to particular master records. For example, the user could navigate through the hierarchy to the Solutions folder beneath a particular service request, click in the list applet, and select New Record from the applet-level menu to associate a solution record from an association applet. The product solution record would become a detail record of the service request.

A tree applet in an explorer view uses the set of master-detail relationships implemented in the business object assigned to the view. As described in [“About Business Objects” on page 418](#), a business object implements a business model or entity-relationship diagram, and specifies the set of master-detail relationships between the business components it includes. This makes it possible to arrange the records of these various business components hierarchically, which can be a very useful feature.

Figure 215 shows the full set of master-detail relationships in the Service Request business object.

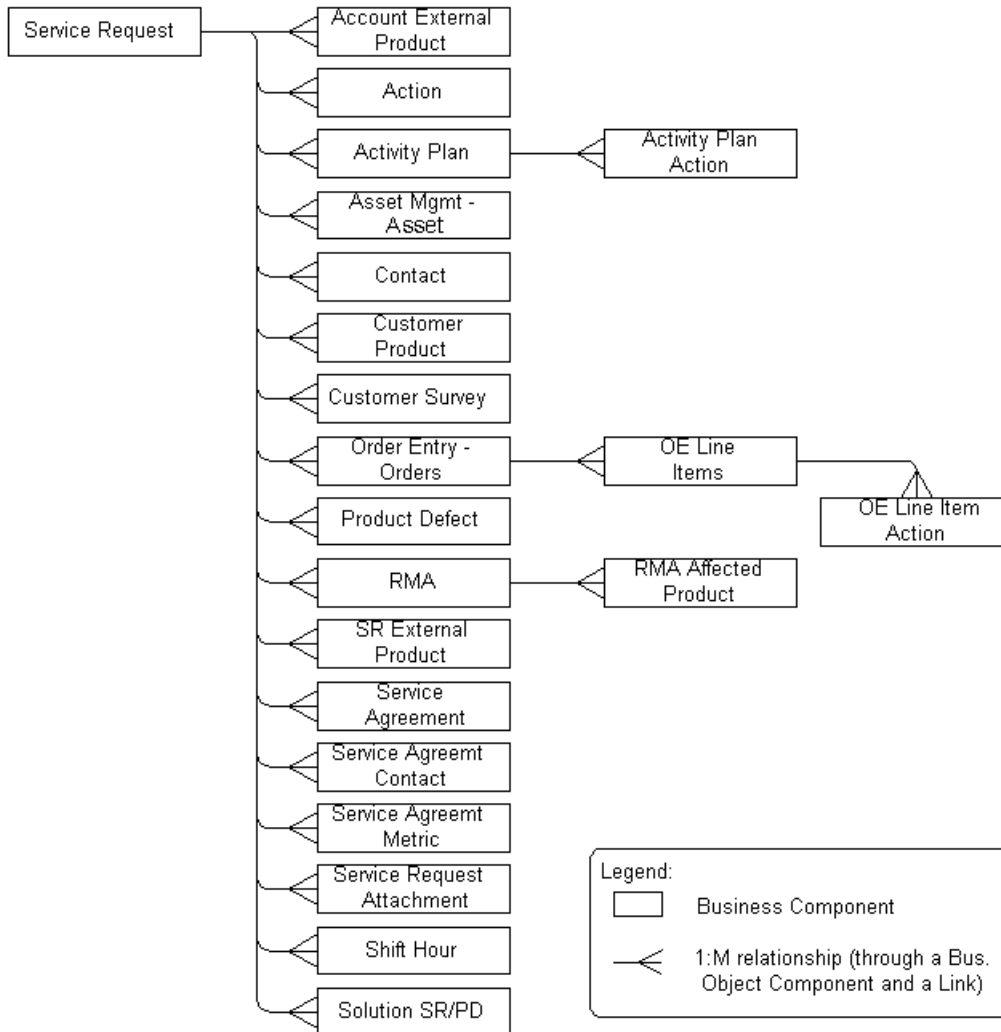


Figure 215. Service Request Business Object

The portion of the Service Request business object used in the Service Request Explorer view is shown in [Figure 216](#).

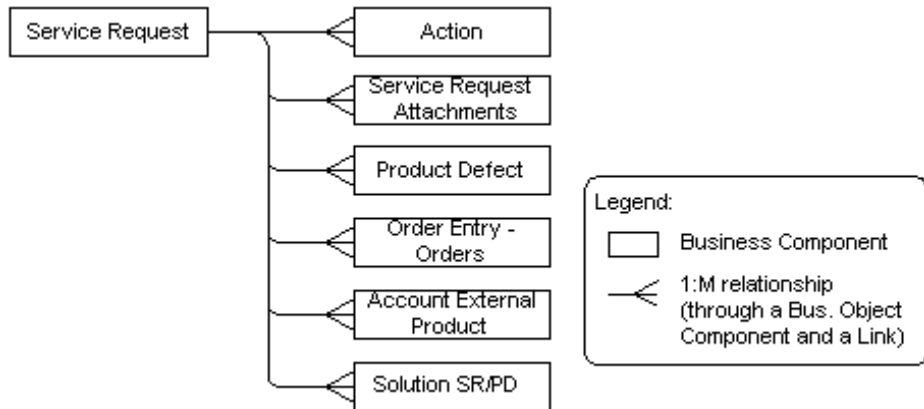


Figure 216. Service Request Business Object Components Used in SR Explorer View

The correspondence between business components in the Service Request business object and folder names in the tree applet is indicated in [Table 51](#).

Table 51. Business Components Corresponding to Folder Names

Business Component	Folder Name in Tree Applet
Service Request	Service Requests
Action	Activities
Service Request Attachment	Attachments
Product Defect	Change Request
Order Entry - Orders	Service Orders
Account External Product	Service Profile
Solution SR/PD	Solutions

The tree applet and explorer view for service requests can be reconfigured to include additional business components. For example, Contacts, Customer Surveys, and Service Agreements folders could be added as child folders of Service Requests, and a Line Items folder could be added as a child of RMAs/Service Orders. However, only business components from the business object (Service Request in this case) can be added in an explorer view based on that business object. Furthermore, a business component can only be added as the immediate child folder of the business component that is its master in the business object. For example, you could add Order Entry Line Items as a child of RMAs/Service Orders, but not of Activities.

Configuring Tree Applets and Explorer Views

A tree applet appears in the left sectors of an explorer view. The applet has a tree object definition as a child. The tree object definition has tree node children. Each tree node child object definition implements one folder symbol. These object types are described in greater detail below.

View

An explorer view has a tree applet mapped to it as a View Web Template Item. However, there are no list applets mapped to it. The list applet is determined dynamically by the folder (Tree Node) that is currently highlighted by the user. A significant property of the View object the Business Object property. The business object selected determines which business components can be displayed, and which business components can be indicated as child nodes of which other nodes.

Tree Applet

A tree applet has no special property settings in the applet object definition, other than the class, which is set to CSSFrameTree. The Applet object type has the following important property settings:

- **Class.** Set to CSSFrameTree. This is required in order for the tree applet functionality to work.

- **Business Component.** Points to the same business component as the top-level tree node.

NOTE: Search specifications on tree applets are not supported.

Tree

The Tree object definition provides only a name; it is an object definition to which tree nodes can be attached, and which itself can be attached to the applet object definition. It always has the name “Tree.” The Tree object type is similar to the List object type used in list applets, in that it serves as an attachment point for child object definitions.

Tree Node

Each folder symbol is implemented using one tree node object definition. This includes the top-level node (Service Requests in the example). All of the tree node object definitions are immediate child object definitions of the tree object definition. There is no hierarchy of child and grandchild tree node object definitions (reflecting the hierarchy in the tree applet) under the tree; this is not feasible in the object definitions hierarchy in the repository. Instead, each tree node’s hierarchical position in the tree applet is specified in the Position property of the tree node object definition.

The Tree Node object definition has the following important properties:

- **Display Name.** This property specifies the name of the tree node (folder) as it will appear in the tree applet in Siebel applications. The display name appears to the immediate right of the folder symbol.
- **Applet.** This property specifies the applet that is opened in the right half of the view when the user opens the corresponding folder. Generally a list applet is specified. The applet must be based on a business component that is in the appropriate hierarchical position in the business object.

- **Position.** The tree node's hierarchical position relative to other tree nodes, and its sequence on its level, are specified with this property. The Position value consists of an integer, or a set of integers separated by periods, such as 1.1.2. The top-level node (Service Requests in the example) is specified as having a position of 1. All immediate child nodes of the top-level node have values of the form 1.x, where x specifies the node's order relative to other nodes on the same level. For example, in order for the Activities folder to appear after the Attachments folder rather than before it, their Position values (1.1 and 1.2, respectively) should be swapped.

To attach a child node at the third level, you specify a Position value for the new node with its first two integers matching the position of the node to attach it to. For example, to attach a node to the RMAs/Service Orders node (currently 1.4), you would give the new node a position of 1.4.1. In general, the rightmost digit in a position specifies its order relative to others on the same level, and all other digits specify the position it attaches to.

- **Business Component.** This needs to be set to the same business component as is specified in the right-side applet.
- **Label Field.** This property points to the name of the field that is used to populate the names in the record list that appears when the node is expanded by the user. For example, the Order Number field would provide the values for the RMAs/Service Orders node, and the Description field for the Activities node.
- **Selected Bitmap Index.** This should be set to the value 5, which corresponds to the folder symbol.

Using the Tree Applet Wizard

The following procedure identifies the steps required to create a new tree applet using the Tree Applet Wizard.

NOTE: The Tree Applet Wizard creates the tree object but does not create Tree Node child objects. You must add a Tree Node object for each applet that you want to appear in the Explorer section of the view, including the top-level node, such as service requests. See [“Tree Node” on page 761](#) for more information.

To create a tree applet using the Tree Applet Wizard

- 1** From the Tools main menu, choose File > New Object.

The New Object Wizard dialog box appears.

- 2** Select the Applets tab and then select the Tree Applet icon.

- 3** In the General dialog box, enter the following information, and then click Next:

- Project
- Business Component
- Name
- Display Name

- 4** In the Web Layout-General dialog box, select the Web template to use for the tree applet, and then click Next.

Some templates used for tree applets are:

- Applet Tree
- Applet Tree 2
- Applet Tree Marketing

- 5** In the Finish dialog box, review the information, and then click Finish.

The Tree Applet Wizard creates the tree object and sets the required properties based on the information you entered.

Tree Applets in the Applet Web Template Layout Window

Tree applets can be created and modified in the Web Layout Editor. When you drag a TreeControl onto the applet, the tree controls and the Tree object definition are created.

When you right-click over the tree control, a pop-up menu appears with the following tree-specific options:

- **Select Tree option.** Allows you to copy and paste the tree control into another applet.
- **Create New Tree Node option.** Adds a new tree node to the tree. The tree node is created at the top level, and is subsequently moved using the Move Selected Tree Node option.
- **Move Selected Tree Node option.** Allows you to change the position of the tree node in the tree. You first click on the tree node you wish to move. Then you can use up, down, left, and right arrow keys, with the SHIFT key depressed, to move the node up or down a level or change its position within its level.

The Position property on all of the nodes is automatically updated for all operations.

Pressing the DELETE key when the tree appears in the Applet Web Template Layout window deletes the currently selected tree node. The Undo and Redo options in the Edit menu are active for all tree manipulation operations in the Applet Web Editor.

Recursive Trees

In a recursive tree, all levels in the hierarchy are of the same object type. For example, the Account Explorer Applet consists of a tree applet in which the only node is for the Account business component, and subaccounts appear beneath accounts which have them. Recursive trees are provided in standard Siebel applications for accounts, activities, campaigns, opportunities, positions, and various other business components in which records can have subrecords. Almost any number of levels of subrecords are possible in a recursive tree.

In order for a recursive tree to be implemented, the business component used must contain a pointer to the record of the same type at the next level up in the hierarchy. In the accounts tree example, the Account business component has a Parent Account Id field which points to its parent account. A Link object definition must exist that references this field in its Destination Field property. In the accounts example, this link is Account/Account.

A recursive tree is implemented with a tree object definition to which only one tree node object definition is attached. In the Tree Node object definition, the following special properties are set:

- **Recursive.** This property is set to TRUE to indicate that this is a recursive tree.
- **Recursive Link.** This property points to the link object definition that specifies the one-to-many relationship between the master business component and itself.
- **Root Search Spec.** This property contains a search specification expression that identifies how the list of top-level records is derived. Generally the top-level records are those that have nothing in the parent Id field, and therefore the search specification is of the form “[Parent xxx Id] is NULL,” for example, “[Parent Account Id] is NULL.”

File Attachment Applets

A *file attachment applet* (or *attachment applet*) provides access to external documents, such as spreadsheets, word processing documents, and slide presentations, from within Siebel applications. A file attachment applet provides the following capabilities:

- Allows the user to open a document of any Windows-supported file type by clicking on its name in a list.
- Allows the user to add document files to a list, edit them, or remove them.
- Provides synchronization and shared access support for attached documents.

An example of a file attachment applet appears in [Figure 217](#).

Attachment Name	Size (In Bytes)	Type	Modified	Update File	Comments
Corp.Slide Pres	2,694,361	ppt	9/10/2001 10:36:34	✓	
Marriott	46,646	pdf	8/16/2001 6:51:29 P		
Siebel Incentive Cont	297,518	doc	9/10/2001 10:36:57	✓	

Figure 217. File Attachment Applet: Account Attachment View

[Figure 217](#) shows the Account Attachment view. The upper applet is the standard Account Form Applet. The lower applet is a file attachment applet called Account Attachment Applet. There is a master-detail relationship between the account and the list of account attachments, so that all file attachments for the current account are listed in the lower applet.

Each document is represented by a row in the attachments list. The document's filename, local/server status, file size, Windows file type (filename extension), and date of last update are displayed. Additionally, the name of each file appears in underlined, colored text, indicating that the file may be opened in the appropriate Windows application by clicking on the name.

To add a document to the attachment list

- 1 On the Attachments list, choose the menu button and choose New Record.

The Attachments list now shows the following:

Menu [v] Save Cancel

File: Browse...

Attachment Name: URL:

Download File:

Update File:

Comments:

- 2 Select the Browse button to display the Choose File dialog box.

The Windows Choose File dialog box appears.

The Siebel application searches for files to be attached in the directory specified in the Start in field of the program icon's Properties dialog (Start > Programs > *Program_Name* > *Program_Icon*). If a user chooses a different folder when attaching a file, the Siebel application searches for the file in this folder the next time the user attaches a file.

Configuring Attachment Applets

Attachment applets use functionality built into the Siebel file system. Various specialized objects and methods are provided in the Siebel File System that provide the attachment support and synchronization capabilities:

- An attachment applet is based on an attachment business component. The details of attachment business components are discussed in the section [“Configuring Attachment Business Components” on page 769](#).
- An attachment applet has the Class property set to either `CSSFrameListFile` or `CSSFrameFile`. `CSSFrameListFile` is used for attachment list applets. `CSSFrameFile` is used for attachment form applets.
- The Name list column or text box control has a Detail Applet property setting of File Popup Applet. This refers to the dialog box that appears when you click on the ellipsis button in the list column or text box.

Several of the list columns or controls in the applet are based on corresponding fields in the attachment business component. This business component is described in [“Configuring Attachment Business Components” on page 769](#). These will typically include those listed in [Table 52](#).

Table 52. List Columns or Controls in an Attachment Applet

Display Name	Field	Type
Name	<code>xxxFileName</code>	TextBox
Local	Dock Status	CheckBox
Request	<code>xxxFileDockReqFlg</code>	CheckBox
Size	<code>xxxFileSize</code>	TextBox
Type	<code>xxxFileExt</code>	TextBox
Modified	<code>xxxFileDate</code>	TextBox
Auto Update	<code>xxxFileAutoUpdFlg</code>	CheckBox

The “xxx” prefix refers to a standard prefix found in the names of the fields in the attachment business component. For example, for account attachments the prefix is “AcCnt” and the actual field names referenced from the applet would be AcCntFileName, AcCntFileDockReqFlg, and so on.

Configuring Attachment Business Components

The Business Component property of the attachment list applet identifies the business component that the Siebel file system uses to store the attachment list data. For the Account Attachment Applet, this business component is called Account Attachment. The attachment business component must adhere to the following requirements:

- The Class property of the Business Component object must be set to CSSBCFile.
- The Table property must refer to an attachment table, as described in the section [“Configuring Attachment Tables” on page 771](#). In the Account Attachment Applet, this table is S_ACCNT_ATT.
- Two Business Component User Prop object definitions must be created as children of the attachment business component, as follows:
 - **DefaultPrefix user.** This is the text of the prefix used in the names of the Siebel File Engine-based field object definitions. These are fields which are based on the base table for the business component. (There may be fields which are based on a joined table, and these will have a different prefix.) For the Account Attachment business component in the example, the prefix is “AcCnt,” which appears in field names such as AcCntFileName and AcCntAutoUpdFlag.
 - **FileMustExist user.** This is a TRUE/FALSE value indicating whether or not the user can enter the name of a file to be provided later. Typically this is set to TRUE, indicating that the file must already exist in order to add it as an attachment.
- The FileDockReqFlg field must have the value for the predefault property set to “N”. The FileDockReqFlg maps to the required column FILE_DOCK_REQ_FLG in the attachment table.

The field names of file engine-supplied fields have to adhere to a special format, and map to specific column names in the attachment table. These names consist of the prefix, as specified in the DefaultPrefix user property, followed by a required set of suffixes. These field names, corresponding columns, and data types are listed in the [Table 53](#).

Table 53. Fields in an Attachment Business Component

Name	Column	Type	Text Length
xxxFileAutoUpdFlg	FILE_AUTO_UPD_FLG	DTYPE_BOOL	1
xxxFileDate	FILE_DATE	DTYPE_DATETIME	
xxxFileDeferFlg	FILE_DEFER_FLG	DTYPE_TEXT	1
xxxFileDockReqFlg	FILE_DOCK_REQ_FLG	DTYPE_TEXT	1
xxxFileDockStatFlg	FILE_DOCK_STAT_FLG	DTYPE_TEXT	1
xxxFileExt	FILE_EXT	DTYPE_TEXT	10
xxxFileName	FILE_NAME	DTYPE_TEXT	220
xxxFileRev	FILE_REV_NUM	DTYPE_ID	15
xxxFileSize	FILE_SIZE	DTYPE_NUMBER	
xxxFileSrcPath	FILE_SRC_PATH	DTYPE_TEXT	220
xxxFileSrcType	FILE_SRC_TYPE	DTYPE_TEXT	30

[Table 54](#) lists a non-file engine field that will usually be present, although it is not required.

Table 54. Non-File Engine Field in an Attachment Business Component

Name	Column	Type	Calculation
Dock Status	(calculated)	DTYPE_BOOL	If ([AcctFileDockStatFlg] = "N" OR [AcctFileDockStatFlg] IS NULL, "N", "Y")

Additional fields can be included as needed. For specialized uses of attachments, such as an image control, the file engine fields may be present in addition to the fields from a standard business component (often through a join). For example, a Product or Literature business component can contain file engine fields to support display of product picture or brochure picture bitmap images.

Multiple sets of file engine fields from different tables can be incorporated in the same business component. For example, literature attachments can have sub-attachments, with the subattachments derived from an intersection table or extension table. The field name prefix must be different for each table.

Configuring Attachment Tables

Attachment tables provide the underlying data storage for the attachment business components. Unlike the attachment business component, which can support purposes in addition to file engine functionality, the attachment table stores file engine data only.

Users will not populate the attachment table directly. Rather, users are provided with an initially empty attachment table, and populate it one file at a time using drag and drop or the browser dialog box in the corresponding file attachment applet.

[Table 55](#) lists the columns that appear in an attachment table. Note that the columns whose names begin with FILE_ are required columns, and must be named as specified in the table. The User Name values can be the same as or different from those listed.

Table 55. File Columns in an Attachment Table

Name	Default	User Name	Type	Physical Type	Length
FILE_AUTO_UPD_FLG		File Auto Upd Flg	Data (Public)	Character	1
FILE_DATE		File Date	Data (Public)	Date Time	
FILE_DEFER_FLG		File Defer Flg	Data (Public)	Character	1
FILE_DOCK_REQ_FLG		File Dock Req Flg	Data (Public)	Character	1
FILE_DOCK_STAT_FLG		File Dock Stat Flg	Data (Public)	Character	1

Table 55. File Columns in an Attachment Table

Name	Default	User Name	Type	Physical Type	Length
FILE_EXT		File Ext	Data (Public)	Varchar	10
FILE_NAME		File Name	Data (Public)	Varchar	255
FILE_REV_NUM	0	File Rev Num	Data (Public)	Varchar	15
FILE_SIZE		File Size	Data (Public)	Number	22
FILE_SRC_PATH		File Src Path	Data (Public)	Varchar	255
FILE_SRC_TYPE		File Src Type	Data (Public)	Varchar	30

Various system columns not related to the file engine will also be present, such as CREATED, LAST_UPD_BY, and ROW_ID.

A table that has file engine columns must be flagged as such with a TRUE value in the File property of the corresponding table object definition.

Pop-Up Windows

This section describes how to create pop-up windows and dialog boxes. There are various scenarios in which pop-up windows are implemented:

- [“Configuring Pop-Up Applets Launched from Applets”](#)
- [“Configuring Pop-Up Wizards” on page 774](#)
- [“Configuring Pop-Up Views Launched from Applets” on page 775](#)

When configuring pop-up windows, consider the following:

- Pop-up applets must use classes derived from CSSSWEFramePopup. Business components are not required for the applets. However, if your pop-up applet is associated with a business component, that business component must be a child of the business object of the view containing the applet from which the pop-up window is launched.

- One level of pop-up window is supported. If you activate a pop-up window from within a pop-up window, it replaces the original pop-up window.
- The More/Less feature is not supported on Pop-up applets. See [“More/Less Mode-Specific Mappings” on page 891](#) for more information.

Configuring Pop-Up Applets Launched from Applets

This is the typical scenario in which clicking a button on an applet invokes a pop-up window for editing a set of values, or browsing through a list, and so on.

To configure a pop-up applet

- 1 Select the applet from which to launch the pop-up window.
- 2 Create a control for the applet.
- 3 Set the Method Invoked property of the control to ShowPopup.
- 4 Expand the Control object, and then select the Control User Prop object.
- 5 Create three control user properties:
 - **Popup.** Set to the applet you want to appear. This applet must use a class derived from CSSSWEFramePopup.
 - **Mode.** Optional. Mode of the applet, either Base or Edit. If not specified, the default is Base.
 - **Popup Dimension.** Optional. Dimension of the pop-up window. The format is Height X Width, for example 500 X 800. If not specified, the dimensions will default to the value specified in the applet’s HTML Popup Dimension property. If that is not specified, the pop-up window dimensions will default to 600 X 600.
- 6 Create the pop-up applet.
- 7 Add controls to the pop-up applet:
 - **Cancel.** Set the Method Invoked property of the control to CloseApplet or UndoRecord. This will close the pop-up applet when Cancel is clicked.

- **OK.** Set the Method Invoked property of the control to CloseApplet to close the applet after you finish processing other calls within your invoked method. This will close the pop-up applet, and then refresh the parent applet in the main browser window.

Configuring Pop-Up Wizards

If you want to have a wizard-style set of pop-up applets, the procedure is similar to configuring a dialog box invoked from an applet control. See [“Configuring Pop-Up Applets Launched from Applets” on page 773](#).

NOTE: The parent applet must be in Edit mode.

To configure a pop-up wizard

- 1** In Siebel Tools, select the pop-up applet, expand the Applet object, and then select the Applet Web Template object.
- 2** Add multiple templates of type Edit in Siebel Tools to the Applet Web Template.
- 3** Assign a different value to the Sequence property of each of the templates, in the order you want them to appear.
- 4** To navigate between pages, add two controls:
 - a** Previous button:
 - Set the Method Invoked property to PostChanges.
 - Add a Control User Prop child object called Sequence with a value of -1.

This posts the changes that the user has made, and then goes back to the page whose sequence number is one less than the current one.
 - b** Next button:
 - Set the Method Invoked property to PostChanges.
 - Add a Control User Prop child object called Sequence with a value of 1.

This posts the changes that the user has made, and then goes to the page whose sequence number is one greater than the current one.

- 5 On the last template, create a control called Finish that closes the applet, and then updates the parent applet.

Configuring Pop-Up Views Launched from Applets

A view (rather than a single applet) can be loaded into a pop-up window. This is not recommended, especially in employee applications. Instead, you should navigate to a new view in the same window (invoking the GotoView method) rather than pop up a view in a new window. However, if there is a requirement for this functionality, perform the following task.

To configure a pop-up view

- 1 Select the applet from which to launch the pop-up view.
- 2 Create a control for the applet.
- 3 Set the Method Invoked property of the control to ShowPopup.
- 4 Expand the Control object, and then select the Control User Prop object.
- 5 Create two control user properties:
 - **View.** Set to the view that you want to pop up.
 - **Popup Dimension.** Set the dimensions of the pop-up. The format is Height X Width, for example 500 X 800.

Users must use the browser's Close (**X**) button to close the window. There is no way to close the window programmatically.

ActiveX Controls

An ActiveX control is a self-contained program unit that can be run from within other programs. An ActiveX control typically registers itself in the Windows registry. In Siebel applications, any registered ActiveX control can be incorporated in an applet. This provides the means to add one or more specialized features to an applet, such as a slider or media player. You can also embed entire applications that are available as ActiveX controls.

NOTE: ActiveX controls will work in most environments, but the programming environment itself may or may not support it. For example, trying to insert a Siebel ActiveX application control into an Excel worksheet generates a “Cannot insert object” error.

Certain third-party ActiveX controls, for example Microsoft Web Browser, Microsoft Rich Textbox, and CTreeView, do not work with Siebel applications and are not supported.

Creating DLL and Class Objects That Reference an ActiveX Control

To make an ActiveX control available for use, you must create DLL and Class objects in Siebel Tools that reference the CAB file containing the control.

To create an ActiveX control

- 1 Create a CAB file containing the control, if it does not already exist.

Microsoft provides some utilities for doing this.

- 2 Copy the CAB file to the correct folder.

- a** When deploying to a server environment, copy the CAB file to the
< *Siebel_install_dir* > \SWEApp\public*language_code*\applets folder,

where:

< *Siebel_install_dir* > is the full path to the Siebel Web Applications installation directory on the Siebel Web Server

language_code is the three-letter code for the language, for example ENU for US English and JPN for Japanese

- b** When deploying a CAB file on the Mobile Web Client, copy the CAB file to the < *Siebel_Client_install_dir* > \PUBLIC*language_code*\applets folder,

where:

< *Siebel_Client_install_dir* > is the full path to the Siebel Mobile Web Client root directory

language_code is the three-letter code for the language, for example ENU for US English and JPN for Japanese

- 3** In Siebel Tools, select the DLL object, and then add a new record.
- 4** Fill in the fields as shown in the following table.

Field	Value
Name	User-defined name for the DLL object
Project	A currently locked project in the Siebel Repository
File Name	File name & version that references the CAB file containing the ActiveX control, for example: subman.cab#Version = 7,0,0,0
Code or Class Id	Class Id of the ActiveX control, for example: clsid:06314967-EECF-11D2-9D64-0000949887BE

- 5** Select the Class object, and then add a new record.

- 6 Fill in the fields as shown in the following table.

Field	Value
Name	User-defined name of the Class object
Project	The locked project used in Step 4
DLL	Name of the DLL object created in Step 4
Object Type	ActiveX Control

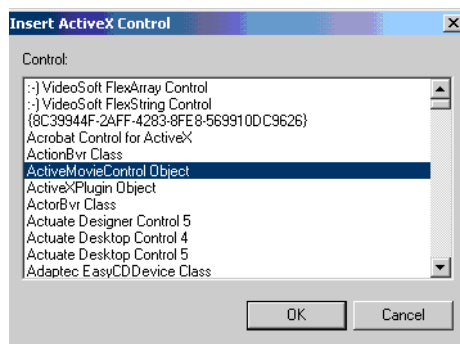
Adding an ActiveX Control to an Applet

You add an ActiveX control to an applet using the Applet Web Editor.

To add an ActiveX control to the applet

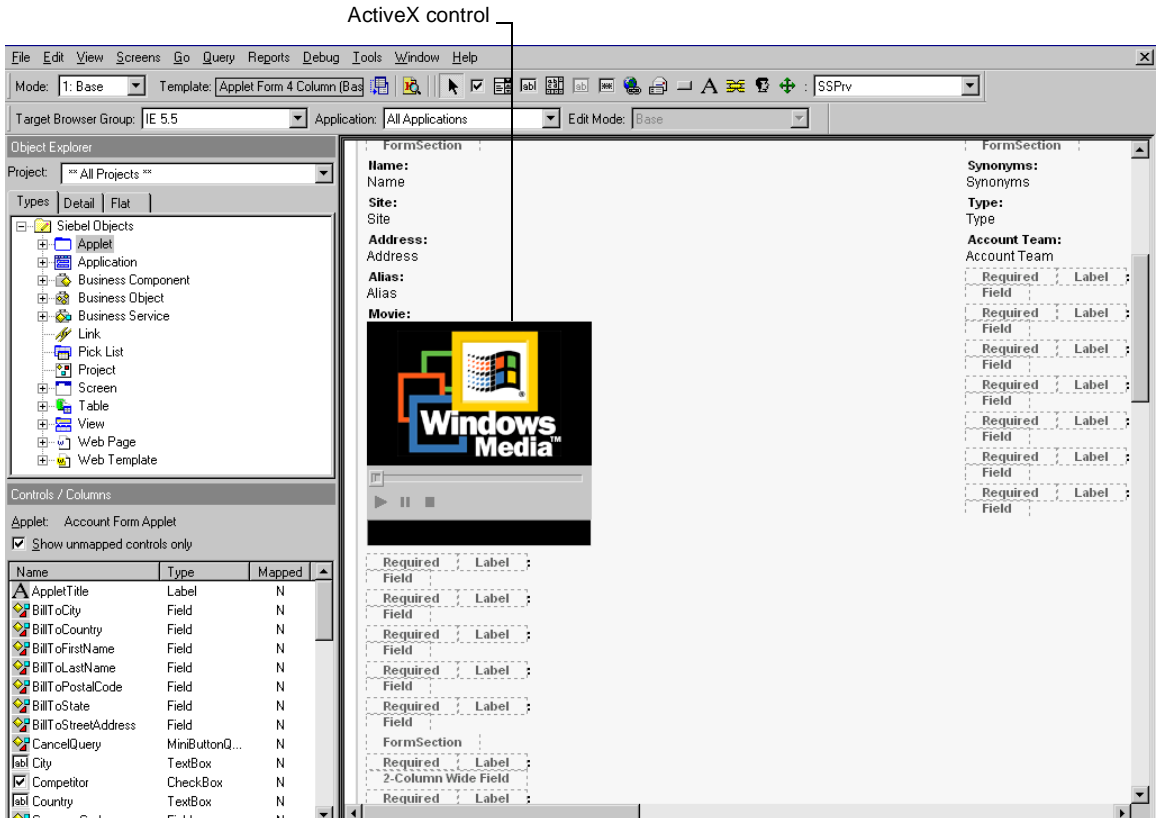
- 1 Open the applet in the Applet Web Editor.
- 2 Click the ActiveX Control toolbar icon in the Web Control toolbar.
- 3 Drag the control to a placeholder in the Web template.

The Insert ActiveX Control dialog box appears for the selection of one of the currently registered ActiveX controls on your system.



- 4 Select the desired ActiveX control and click OK.

The selected control replaces the placeholder in the Applet Web Editor.



The illustration shows a movie player control called ActiveMovie Control in a form applet.

- 5 Set the Class property of the control to the name of the class created in [Step 6](#) of “[To create an ActiveX control.](#)”

The default properties of the ActiveX control will be defined as Control User Properties.

- 6 Compile the .srf file.

Setting Properties in an ActiveX Control

An ActiveX control includes its own property list, which varies from control to control. In addition, an ActiveX control in an applet has the full set of properties of the Control object type. There are two ways to view and modify properties for an ActiveX control in an applet: by using the Properties window, or by activating the control's built-in property sheet.

To change properties in the Properties window

- 1 Choose View > Windows Properties.
- 2 Select the ActiveX control in the Applet Web Template Layout window.

The Properties window lists the properties for the ActiveX control.

- 3 Click the Categorized tab at the top of the Properties window.

This clusters all of the control's native properties under the ActiveX heading, and all of the standard Control object type properties under the Misc. heading.

- 4 Make changes to property settings as you would in any Siebel object definition.

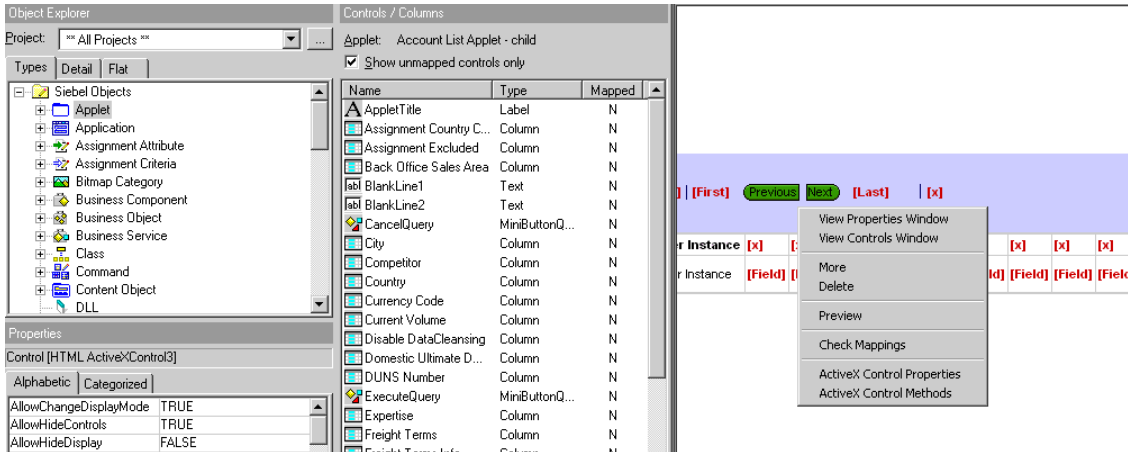
The changes you make to the control's native properties are generally displayed in the Applet Web Template Layout window, such as when you change a text color or font property. Additionally, changes you make to the control's native properties are saved with the applet, just as with the Siebel properties.

The alternative approach to changing property settings is to use the control's native property sheet. When you make property changes using the control's native property sheet, you can modify only the properties of the ActiveX object, not those of the standard control object definition. To alter the standard control properties, you must use the Properties window.

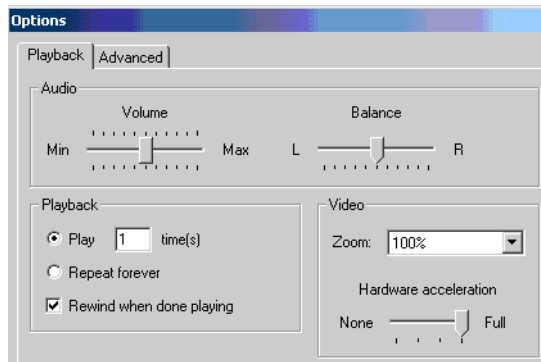
To change properties in the ActiveX control's native property sheet

- 1 Select the ActiveX control in the Applet Web Template Layout window, and right-click.

- In the shortcut menu that appears, select ActiveX Control Properties.



If the control has a native property sheet, it is activated. The native property sheet for the ActiveMovie control appears below:



- Make any desired changes to property settings. These settings are saved with the applet when you exit the Applet Web Template Layout window or do a Save.

ActiveX Methods and Events

An ActiveX control in an applet exposes a set of methods and events that are provided with the control. The methods may be called from scripts written in browser script attached to the control or other objects, and event procedures can be programmed in to respond to the events the control generates.

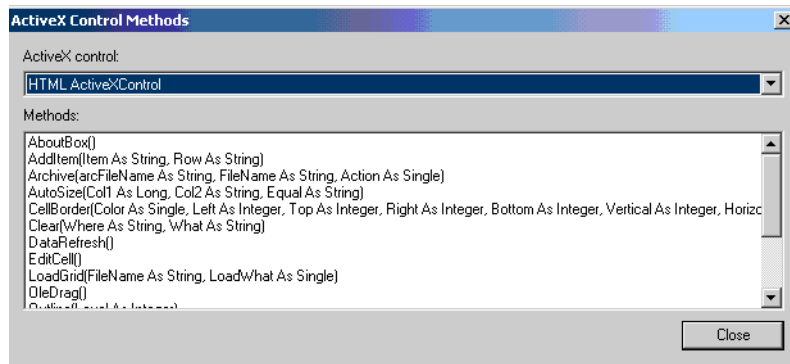
To see the list of available methods for the ActiveX control

- 1 In the Applet Web Template Layout window, click the ActiveX control to select it.
- 2 Right-click to access the shortcut menu.

See [“To change properties in the ActiveX control’s native property sheet” on page 780](#) for an illustration of the shortcut menu.

- 3 Select the ActiveX Control Methods menu option.

The ActiveX Control Methods dialog box appears.



This dialog box lists the methods, and specifies the syntax for calling them. It is for reference purposes only.

- 4 Click Close to dismiss the dialog box.

Distributing ActiveX Controls

You can distribute ActiveX controls to licensed end users. When you do this, you will also need to distribute any dependency DLLs along with the ActiveX controls. For information about these dependency DLLs (such as the number and type of DLLs that need to accompany the ActiveX controls), contact the ActiveX vendor.

HTML Content Controls

HTML content controls allow you to display HTML content in fields in the user interface. The HTML content can be static HTML or HTML from an external content source. You can configure any control type to display HTML content, as shown in [Figure 218](#).



Figure 218. Example Fields Using HTML Content Controls

Configuring fields to display HTML content is a matter of setting properties of the control in Siebel Tools and defining any necessary supporting information, such as URL, host name, syntax, and authentication parameters using a set of administration views in the Siebel Web client.

Control Properties

The key Control object type properties for displaying HTML content are the following:

- **Field Retrieval Type.** This property determines the type of HTML to be displayed in the field. Possible values are:
 - **Symbolic URL.** This value specifies that the content will come from an external host based on a symbolic URL. You need to define the necessary information needed to access the external source. This includes the syntax used for the request, the host name, necessary arguments, and so on.

See *Siebel Portal Framework Guide* for more information about defining Symbolic URLs.

- **Field Data.** This value specifies that the HTML content will be stored as data.
- **Service.** This value specifies that the field will be rendered by a business service. The control must have a User Prop defined with the name *Field Retrieval Service* and the value is the name of the business service.

For example, you can define a control to display a Content Center asset by setting the Field Retrieval Type to Service and then adding a Control User Property child object with the name Field Retrieval Service and the value ContentBase - Asset Publish Service.

For more information about Content Center Assets, see *Applications Administration Guide*.

- **HTML Attachment.** This value specifies that the field will display an HTML attachment. The control will render the HTML Attachment identified by the underlying field.
- **URL.** This value specifies that content will be displayed from an external source based on the simple URL specified in the underlying field.
- **ContentFixupName.** This property determines how to correct links post processing. It provides the name of a Fixup as displayed in the Fixup Administration View. This value does not work if Field Retrieval Type is HTML Attachment or Service.

- **HTML Display Mode.** This property should be set to DontEncodeData so that the HTML content renders properly in the browser. Possible values are:
 - **DontEncodeData.** Use this value when the field value is actual HTML text and you want it to be shown as such.
 - **EncodeData.** If the field value contains HTML reserved characters, such as angle brackets (< >), ampersand (&), and so on, they are encoded before they are displayed so that they appear correctly within the browser.

Administration Views

There are four administration views that allow you to specify the necessary information related to HTML content controls. This includes information such as, host name, URL, required arguments, authentication parameters, and so on.

The views used to enter this information are described in [Table 56](#):

Table 56. Administration Views for HTML Content

View Name	Menu Path	Description
Host Administration	From the application-level menu, choose View > Site Map > Integration > Host Administration.	Allows you to enter the HTTP host, including virtual name and authentication parameters.
Fixup Administration	From the application-level menu, choose View > Site Map > Integration > Fixup Administration.	Allows you to specify how to handle links post processing.
Symbolic URL Administration	From the application-level menu, choose View > Site Map > Integration > Fixup Administration.	Allows you to specify the content agent for an external host. This includes URL, host name, fixup name, and arguments.
Content Sets	From the application-level menu, choose View > Site Map > Content Center > Content Sets.	Allows you to upload and manage Web content to be rendered in the Siebel Application.

For an overview of content agents and symbolic URLs as well as procedure for administering content agents, see *Siebel Portal Framework Guide*.

The Host Administration View

The Host Administration view is used to specify hosts that either require fixup processing, authentication, or to simply obscure the true host name. Only links associated with a specified host are fixed up.

For each host, you need to specify an external content host server. The specification of a host enables one or more of the following features:

- It obscures the true servername in the generated HTML.
- It allows the specification of a set of NCSA Basic Authentication credentials for content hosts that require authentication.
- It allows administrators to control fixup at the host level.

Fixup Administration View

Administrators can use this view to control the behavior of links embedded within the external content. A fixup has a Link Context, which corresponds to the fixup type. There are four types of fixups:

- **Do Nothing.** This fixup does not affect any of the links. The links (relative or absolute) remain as they are with the content being passed back in its original form.
- **Outside Application.** This fixup converts all of the relative links to absolute links using the host and path of the parent URL. No links are proxied.
- **Inside Application.** This fixup converts all of the relative links to absolute links and any links using a host from the Hosts table (for example, navigated to by choosing Integration Administration > Host Administration) are proxied in order to maintain all of the SWE context.
- **Inside Applet.** This fixup performs the same as the Inside Application fixup.

NOTE: Fixup is required for all links within applications that use high interactivity.

Proxied Links

When using either the Inside Application or Inside Applet fixup type, any link using a host from the Hosts table is proxied. Any relative link is first converted to an absolute URL, and then if the host is in the Hosts table, the link is proxied.

Default Link Targets

There are no default link targets applied to a fixup. However, a fixup may have a link target specified for it, in which case the link target will be added to the fixup.

Configuring Fields to Use Web Content Assets

Any business component can take advantage of Web Content Assets to add fields that are rendered as HTML content. For example, you can use display static HTML messages to your end users in the Partner Relationship Manager application, or represent a Product Description as HTML content.

This section describes the steps for configuring this functionality, using the Partner Message business component as an example.

To configure fields to use Web Content Assets

- 1** In Siebel Tools navigate to the Partner Message business component.
- 2** Navigate to the Message field and set the Pick List to ContentBase Asset Hierarchical PickList.
- 3** Query for Partner Message List Applet.
- 4** Navigate to the list column named Message Body and set the Pick Applet property to ContentBase Asset Hierarchical PickList.
- 5** Query for Partner Message Entry Form Applet.
- 6** Navigate to the control name Message Body Preview and set the Field Retrieval Type property to Service.
- 7** Navigate to the Control User Props for this control and add a Control User Prop with the name Field Retrieval Service and the value ContentBase - Asset Publish Service.
- 8** Query for Partner Message Form Applet (SCW).

- 9** Navigate to the control named `MessageBody` and set the `Field Retrieval Type` property to `Service`.
- 10** Navigate to the `Control User Props` child object for this control and add a control user property with the name `Field Retrieval Service` and the value `ContentBase - Asset Publish Service`.
- 11** Compile your changes and test.

Physical User Interface Layer **14**

This chapter provides an overview of the templates and tags that represent the physical UI for employee and customer/partner applications. These are different from the object definitions in the repository that refer to the logical UI, business objects, and data model layers.

This chapter covers the following topics:

- Understanding Siebel templates
- Understanding Siebel tags
- Navigational constructs in the Siebel Web Client
- Application appearance using standard and high interactivity

For more information on templates and tags, see *Siebel Developer's Reference*.

Understanding Siebel Templates

A Web template is a special kind of file that contains markup (HTML, WML XML, and so on) interspersed with special Siebel Web engine-specific tags (prefixed by “swe”). It defines the layout and formatting of elements of the user interface (such as views, applets, and controls). Web browsers require HTML to define the layout and formatting of a page. Siebel Web Templates provide this HTML layout information to the Siebel Web Engine when rendering Siebel objects in the repository definition of the application. Wireless applications are rendered in the same manner except for the fact that the markup language in the templates is WML or XML. This section focuses on the configuration of Web (HTML) applications, but many of the concepts are generic across markup languages.

Templates are filled with data and user interface elements by associating views, applets, controls, and other objects defined in Siebel Tools with them. For each view, applet, or control, you map the repository definition to a placeholder in the template. For example, you may have a View object with three applets. You associate a View Template with the view, and map each applet to a placeholder in that template.

An important feature of Siebel Web Templates is that they can be shared between many objects in the repository. Because a template has only placeholders, any number of repository objects can be mapped to a specific placeholder. This allows you to propagate style or structural changes to numerous user interface elements by changing only one template. A typical Web application will contain on the order of 5-50 templates, which together form the bases for several hundred views and applets. For instance, a template which defines the layout and formatting of a standard list applet can be shared among all list applets repository definitions in an application.

The reusability of templates is further enhanced in that the Siebel Web Engine skips over template placeholders which are not mapped in the repository. If a placeholder is not mapped, then it and the HTML contained in between the Siebel tags that define the placeholder are simply ignored. Thus, if the template contains layout for a 10 column wide list applet, but only 2 of the columns are mapped, the other 8 are simply ignored.

Siebel Applications provide numerous applet and view templates with the product, which are extremely flexible; you may not have to modify any of the applet and view templates to support your migrated application. However, in some cases (especially customer and partner applications) you may wish to modify the default templates to reflect your corporate look and feel, or, in some cases, create an entirely new template. Siebel templates must use valid HTML. Adding JavaScript beyond what is already generated by the Siebel Web Engine is not recommended. If it is necessary to add JavaScript, it should be done in Siebel Tools using Browser Script.

NOTE: You can view Web templates using Siebel Tools, but you modify templates using an external editor. For more information, see [“Web Template Explorer” on page 794](#) and [“Setting Web Preferences” on page 795](#).

To allow for even greater flexibility, Siebel eBusiness Applications have provided a mechanism in which a particular template file can include another one. This device is used, for example, to separate handling of the title of an applet from the body. A standard applet layout can be defined once and combined with multiple different title layouts by including a template file that defines the title within the applet template.

By convention, the filenames of Siebel Templates take the .SWT extension; for example, CCPageContainer.SWT, CCHomePageView.SWT, and so on. This Siebel-suggested convention is an abbreviation for Siebel Web Template.

You do not have to follow this convention; the Siebel Web Engine recognizes and interprets the files correctly regardless of how you name them. However, ending your Siebel filenames with .SWT may help you.

NOTE: Template files are typically stored in the Web Template directory under your Tools installation directory. The Filename property references the Web Template object type.

The layout and style of HTML Web pages is dynamic, which allows simultaneous support for multiple browser types and versions. Siebel Web templates support conditional branching. Conditions are evaluated based on the results of a business service. See [“Displaying Server Side Errors” on page 911 in Chapter 16, “Special Behavior Supported by Templates.”](#)

Generated HTML Files

After you configure your application and deploy it on your Web site, it becomes available for viewing through a client’s browser.

When a client requests a specific view (either through the application URL directly or by clicking the appropriate link from within another page), the Siebel Web Engine does the following:

- 1** Retrieves the object definition of the view from the .srf and retrieves the object definition of each applet in that view.
- 2** Retrieves the data specified in the object definition from the data manager layer of the Application Object Manager.

- 3 Matches this data with the template specified by the view and each applet within it.
- 4 Renders this view by using the placeholders in the template to define where each element (control, list element) in the object definition is to be placed and how it should be formatted.

When the user views the generated HTML file in a Web browser, it is rendered as a Web Page, and includes all the layout specified in the original template as well as the data and controls retrieved.

Figure 219 shows how Siebel Web Engine generates HTML output using templates, repository definitions, and HTML.

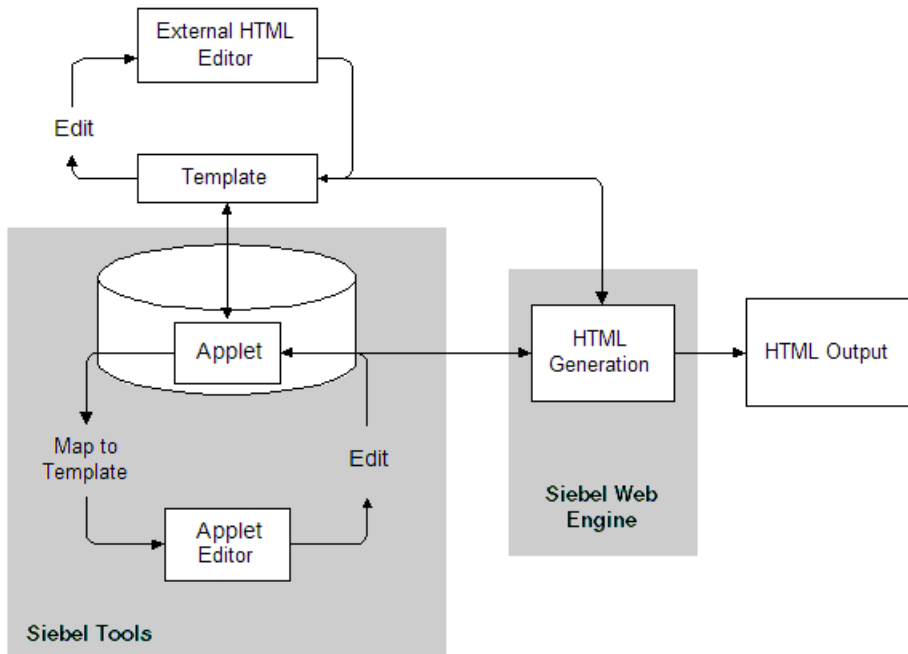


Figure 219. How Siebel Web Engine Generates HTML Output

Types of Templates

The templates you modify fall into one of several groups, depending on the purpose of the template or what the template contains.

- **View template.** Used for displaying a view; specifies where to lay out applets and other page-level controls on the view, and what the formatting of the view should be.
- **Applet template.** Specifies where to lay out fields and controls for an applet. Also specifies the formatting for elements within the applet.

Applets can have more than one mode. The types of modes are:

- **Base:** Read-only mode for displaying but not editing data. Views appear by default in Base mode.
- **Edit.** Mode for editing an existing record.

If New/Query templates do not exist, Edit is used when creating and querying.

- **Edit List.** Allows users to edit fields in a list applet.

If Edit/New/Query templates do not exist, Edit List is used in employee applications running high interactivity when editing, creating, and querying. Standard interactivity applications, such as customer and partner applications, do not use Edit List, so you must define an Edit mode template.

- **New:** Mode for creating a new record.
- **Query:** Mode that allows you to perform a query-by-example (QBE).

Each mode has a corresponding template. Some templates, such as edit-mode templates, can be shared by many applets.

NOTE: New and Query should only be used if they are different from Edit. Otherwise Edit is used.

- **Web Page template.** Specifies the layout of the whole display. Has information about where the screen bar/view bar/view should appear.

- **Page Container template.** Used as container pages for view templates. The overall purpose of the page container is to provide a structure for the overall application. There is one page container per application, but views can be flagged indicating they should not use the container page (for example, the login page cannot use the page container).
- **Formatting templates.** Templates that allow you to create custom HTML types, such as specialized controls, list items, and page items. These templates have the extension .SWF (Siebel Web Format). For more information about .SWF files, refer to [“Adding Sorting Capabilities to Your Application” on page 913 in Chapter 16, “Special Behavior Supported by Templates.”](#)

Your application can contain other pages, of course, that do not contain any Siebel tags. For example, you may have an About This Application help page. However, this page, by definition, is not a template.

Web Template Explorer

The Web Template Explorer window is a Windows Explorer-like concatenated tree listing of Web templates. It allows you to visualize a color-coded HTML view of the Web Template hierarchy. Clicking on the items in the Web Template Explorer displays the HTML source code of the Siebel Web Template (.swt) file for review or editing in the main window. Since a template (parent) can include other templates (children), a split view is presented to view a parent and child template at the same time. The Web Template combo box in the Web Template window allows you the choice of displaying all Web Templates in the Explorer, the top level Web Templates only, or individual Web Templates.

To access the Web Template Explorer

- 1 Choose View > Windows > Web Templates Window.

The Web Template Explorer appears.

- 2 Select a template file in the Web Template Explorer.

The selected Web template appears. If a top-level template is selected, then the parent template is shown in the upper part of the window, while the selected (child) template is shown in the bottom part. The vertical splitter that divides the two parts of the window can be dragged to change the viewable area in each part of the window.

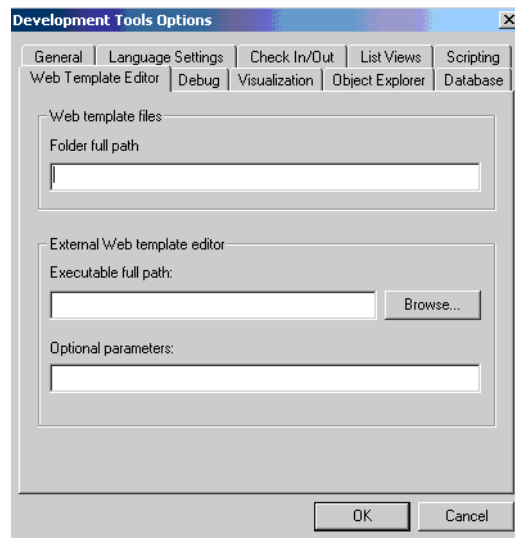
Setting Web Preferences

You can select the HTML editor that should be launched for editing templates by specifying it in the Web Template Editor tab of the Development Tools Options box.

To set Web preferences

- 1 Choose View > Options.
- 2 Select the Web Template Editor tab.

The Web Template Editor tab option appears.



- 3 Specify the path of the external HTML editor executable that you want to use to edit the templates. You can also specify any parameters specific to the command-line invocation of the external editor.
- 4 Click OK.

The changes will take effect the next time the Web Applet or Web View Editor is launched.

Understanding Siebel Tags

Siebel tags are special tags you insert into template files. They specify how objects defined in the repository should be laid out and formatted in the final HTML page in the user's Web browser.

The process of configuring a Web application separates the layout and formatting from the application definition and binding to data. You use Siebel tags to determine the layout and formatting of controls in your application.

Mappings Between Controls and IDs

The .SWT template files do not include references to specific controls in the repository. Instead, they specify a layout and style, with placeholder tags. The following is an example of a Siebel tag that places a Web Page Item in a Web Page. Other Siebel tags might place other items, such as view bars, applets, or controls, for instance, in a Web Page.

```
<swe:Control id="1" property="FormattedHtml" />
```

To process this tag and generate the final HTML the Siebel Web Engine does the following:

- 1 Examines the compiled .srf file for the properties of the Web Page Item in the current Web Page that has an Item Identifier equal to 1. This is the mapping between the template file object and repository object.
- 2 Renders the Formatted HTML representation of this repository object in place of the abstract placeholder in the template file.

Figure 220 shows how the mappings between controls and IDs work for displaying an image as a link to add a new contact. The example illustrates the general point, but note that in your actual implementation, the HREF probably will not look exactly like this. If you create the right controls and template mappings, Siebel Web Engine will construct a URL in the HREF that will execute the method `NewRecord` in the correct context.

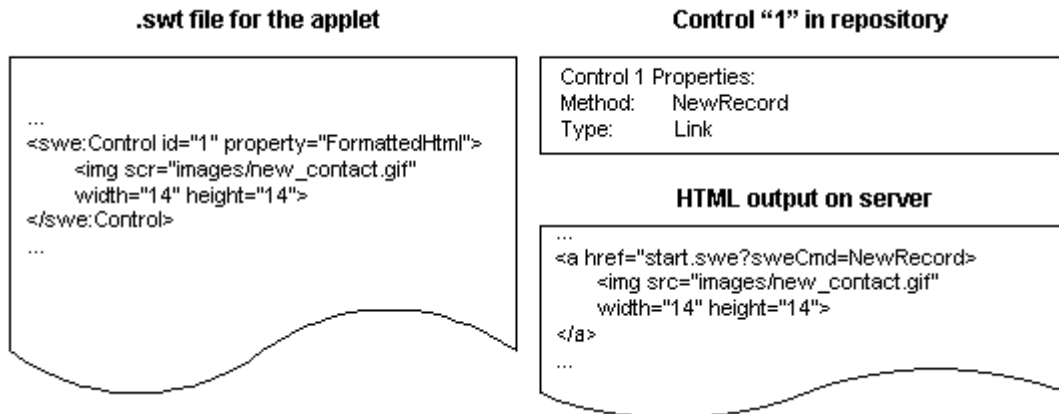


Figure 220. Mappings Between Controls and IDs

Singleton and Multi-Part Tags

Singleton and multi-part tags are part of the basic vocabulary of SGML, so they are only discussed here to introduce the concepts and terminology. Siebel eBusiness Applications use singleton and multi-part tags in the standard way.

A *singleton* element is a tag that includes the end-tag slash in the same tag as the tag name. There are no child elements in a singleton tag. The following is an example of a singleton tag:

```
<swe:pageitem name="value" />
```

The following is an example of a multi-part tag because it does not have the end-tag slash:

```
<swe:control id="1" property="formattedHTML">
    ...HTML here...
</swe:control>
```

“This” Tag

Sometimes you will want to use a multi-part tag, but make reference to the SWE-generated control at some point other than the beginning and end of the tag. To do this, you can use a “this” tag:

```
<swe:control id="1">
    ...HTML here...
    <swe:this property="formattedHTML"/>
</swe:control>
```

The `<swe:this>` tag is an alias for the nearest enclosing Siebel context. Often, this context is established by an enclosing `<swe:xxx>` element. For example, `<swe:this>` commonly appears inside a multi-part `<swe:control>` element. In that case, `<swe:this>` is an alias for the control. It is used to display properties of the control. In some cases, the context is less direct. For example, if an `<swe:this>` element appears in an applet template file, outside of any `<swe:control>` tag, it is an alias for the applet, and can be used to display properties of the applet.

Iterators

Iterator tags specify the number of times the tag should iterate its contents. For example, the `swe:for-each` tag allows you to reduce the size of the template files where the same HTML and Siebel tags are used with controls or page items with different values for the id parameter:

```
<swe:for-each count="x" iteratorName="yyyy" startValue="z"/>
```

Other iterator tags include `swe:for-each-row`, `swe:for-each-child`, `swe:for-each-node`, `swe:for-each-indent`, `swe:for-each-value`.

The attributes of the `swe:for-each` tag are as follows:

- **count.** Specifies the number of times the tag should iterate its contents.

- **startValue.** The value that should be assigned to the iterator at the start of the iteration. The tag will start the iteration by assigning this value to the iterator, and will increment it by one for each iteration.
- **iteratorName.** The name of the iterator. This name can be used to get the value of the iterator during the iteration using the syntax `swe:iteratorName`.

In the section enclosed by the `swe:for-each` tag, references to the current value of the iterator is through the name specified in the `iteratorName` attribute. For example, if you set the value of the `iteratorName` to “CurrentID,” then you can get the value of the iterator using the syntax `swe:CurrentID`. You can also reference a value that is an increment over the current value as `swe:CurrentID+x`. The fragment below illustrates this usage:

```
<swe:for-each startValue="2301" count="50"
iteratorName="currentId">
  <swe:control id="swe:currentId">
    .
    .
  </swe:control>
  <swe:control id="swe:currentId+100" />
</swe:for-each>
```

Nesting and Siebel Tags

You cannot nest a Siebel tag inside HTML tags. For example, the following is not valid and will generate an error:

```
">
```

In addition, you cannot nest some Siebel tags. For example, the following is not valid and will generate an error:

```
<swe:control id="1">  
  <swe:control id="2" property="formattedHTML"/>  
  <swe:this property="formattedHTML"/>  
</swe:control>  
</swe:control>
```

SWE Conditional Tags

The SWE framework supports the `<swe:if>` conditional tag, which provides a simple conditional branching capability. The `<swe:if-var>` tag is a variation on `<swe:if>` that permits you to evaluate a namespace within an applet template. For further information see [Chapter 16, “Special Behavior Supported by Templates.”](#)

High Interactivity Versus Standard Interactivity

Traditional Web applications follow a model whereby almost every user action results in a page refresh. Some of the user actions that can trigger a page refresh are a user changing the quantity of an item in the Siebel eSales Shopping Cart, a user inserting a new appointment in the calendar, and a user selecting a different item from a list to see its details. These frequent page refreshes not only slow down users by forcing them to wait for new pages, but also waste time as users reorient themselves with the frequently changing context caused by these page refreshes. In addition, frequent page refreshes are expensive in terms of network bandwidth utilization, as each page refresh requires the same HTML information, already displayed in the browser, to be downloaded with new data.

High interactivity solves the problem of lowered employee productivity and high bandwidth requirements by reducing the number of page refreshes.

High interactivity depends on capabilities that are only available in Internet Explorer 5.5 and higher versions, and is only used for employee applications such as Siebel Sales and Siebel Call Center. Customer applications such as Siebel eSales and Siebel eService do not use high interactivity.

Some differences between standard interactivity and high interactivity are as follows:

- **Support for client-side scripting.** Client-side scripting is available for both types of applications. However, in high interactivity, customers have access to Siebel objects through which they can build data validation logic on the client side to reduce further the number of page refreshes needed.
- **Support for interactive controls.** High interactivity employs specialized JavaScript controls for drop-downs, date and time, lists, and so on. These controls provide greater levels of interactivity than traditional HTML controls that appear in standard interactivity, are designed for JavaScript controls, and are different from the HTML controls based on list applets used in standard interactivity. For example, the list control supports resizing of columns, and drop-down lists support auto-completion.
- **Support for application-level menus.** Application-level menus require support for Java applets. Because there is no support for Java applets in standard interactivity, there are no application-level menus.
- **Support for extensible toolbars.** You can take advantage of the functionality of high interactivity by extending JavaScript toolbars and creating new ones. JavaScript toolbar objects reside in the JSSApplication hidden frame, which usually does not reload during the application life cycle. Therefore, they are not redrawn when there is a page refresh. The UI part of the JavaScript toolbar resides in a visible HTML frame (recommended to be a persistent frame that reloads infrequently) and redraws when the HTML frame reloads.
- **Support for implicit save.** High interactivity supports an implicit save model whereby navigating off a record causes the changes to be saved. The benefit of this model is the efficiency with which data can be entered; an explicit save operation for each record is not required to commit the changes.

- **Appearance of applet-level menus.** In standard interactivity, the applet-level menus for Siebel applications are represented in the form of drop-down lists. These menus show up as dynamic drop-down controls in high interactivity.
- **Support for browser back and forward buttons.** Standard interactivity supports browser forward and back buttons, which are used for navigating within an application. High interactivity uses Siebel bookmarks to support navigation that is accessible through provided controls for back and forward movement within a session and a history drop-down list.

For more information on high interactivity, its architecture, and enabling it, see [Chapter 2, “Siebel Architecture \(Basic Concepts\).”](#)

Navigational Constructs

The user interface has four navigational constructs that allow you to access screens and views.

- 1 First Navigational Level (Primary Tabs or Screens).** The tab bar consists of tabs allowing you to select a Screen. By using one of these tabs you can navigate to that Screen's default View. Refer to [“Primary Tab Bar” on page 803.](#)
- 2 Second Navigational Level (Visibility picklist, Show picklist, or Context Views picklist).** The Visibility picklist allows you to select a specific View within the current primary tab (Screen). The set of Views listed in this picklist are not all of the Views in the Screen; rather they are the *context views*, generally those Views that have visibility defined, such as My Accounts or My Team's Opportunities. Selection of a context View in the second-level picklist establishes visibility for the Views obtainable from the third-level tabs. Refer to [“Visibility Picklist and Detail Tab View Bar” on page 804.](#)
- 3 Third Navigational Level (Detail View Tabs).** Detail View tabs allow you to navigate from the context View initially selected in the Visibility picklist to a non-context View. This tab bar resides at the top of the detail applet. These tabs provide a form of drilldown: when the user selects a record in the upper applet and chooses a detail View tab, the application navigates to a new View in which the master record is the one selected in the previous view. Refer to [“Visibility Picklist and Detail Tab View Bar” on page 804.](#)

- 4 Fourth Navigational Level (Subcategory Views Picklist, Current View Picklist).** The subcategory views (fourth navigational level) picklist appears inside an applet, just beneath the row of third-level view drilldown tabs. It allows the user to choose a view from the category represented by the selected third-level view bar tab, when the view bar tab is for a category rather than an individual view. Refer to [“Subcategory Views” on page 809](#).

Primary Tab Bar

The tab bar allows you to select between the functional areas of an application (Screens). Selecting a screen by clicking on one of these tabs allows you to navigate to that Screen's default View, and alters the contents of the Visibility picklist by listing context Views for the selected Screen.

The primary tab bar is defined in Tools by using the object definitions of the Page Tab object type. The Page Tab object type is a child of the Application object type. One Page Tab child object definition is specified for each Screen (primary tab) that appears in the primary tab bar, given a particular parent Application object definition. The Page Tab object definition creates an association between a Screen object definition and an Application object definition. For example, the Siebel Service application has primary tabs of Activities, Category, Contacts, Correspondence, and so on, and this is defined by creating Page Tab children of the Application object definition called Siebel Service, one for each tab required.

The primary tab bar or “screenbar” is defined using the `<swe:screenbar>` and embedded `<swe:screenlink>` tags.

The following code is some sample code from a page container template:

```
<tr>
<swe:screenbar>
  <swe:screenlink state="Active" property="FormattedHtml" >
    <td></td>
    <td background="images/nav/tabon_back.gif">
      <nobr>&nbsp;  <swe:screenname/>&nbsp;  </nobr>
    </td>
```

```
<td> </td>

</swe:screenlink>

<swe:screenlink state="Inactive" property="FormattedHtml" >

<td class='tier1Off'><nobr>&nbsp;<swe:screenname />&nbsp;</td>
</nobr></td>

</swe:screenlink>

</swe:screenbar>

<td class="tier1Back"></td>

</tr>
```

The `<swe:screenbar>` tag expands to show all the Page Tab names for Screens assigned to an application. The `<swe:screenlink>` tag defines the link to access for a particular Page Tab. The attribute “state” allows you to specify a different look for when the screenbar is selected (Active) versus when it is not selected (Inactive).

Visibility Picklist and Detail Tab View Bar

Views are grouped into context Views (second level navigation: the Visibility picklist) and non-context Views (third level navigation: the detail View bar). You should choose an initial View that establishes business object and visibility context before drilling down into related Views. The visibility setting that has been established by choosing the context View in the Visibility picklist is maintained during subsequent navigation among the non-context Views available in the detail view bar.

Visibility (specifically in this case, record access visibility) refers to the user's access rights to see particular records of particular business components, as determined from the user's logon and other information maintained by an administrator. For more information on visibility, see *Security Guide for Siebel eBusiness Applications*.

The set of context Views for a selected Screen is established based on the following rules:

- When all Views assigned to the Screen (through Screen View object definitions) belong to the same business object, the context Views are the set of Views in the Screen that have visibility rules (like “My Accounts”, “All Contacts,” and so on). The remaining Views that do not have visibility rules are grouped as non-context views (like “Account-Contacts”, “Charts,” and so on).
- When some of the Views assigned to the Screen belong to different business objects, the entire set of Views in the Screen are grouped as context Views, and appear in the Visibility picklist. Such Screens do not have any non-context Views, and so the detail View bar is empty. An example is the Administration screen.

The segregation of Views into context and non-context views for the Accounts screen is illustrated in [Figure 221](#).

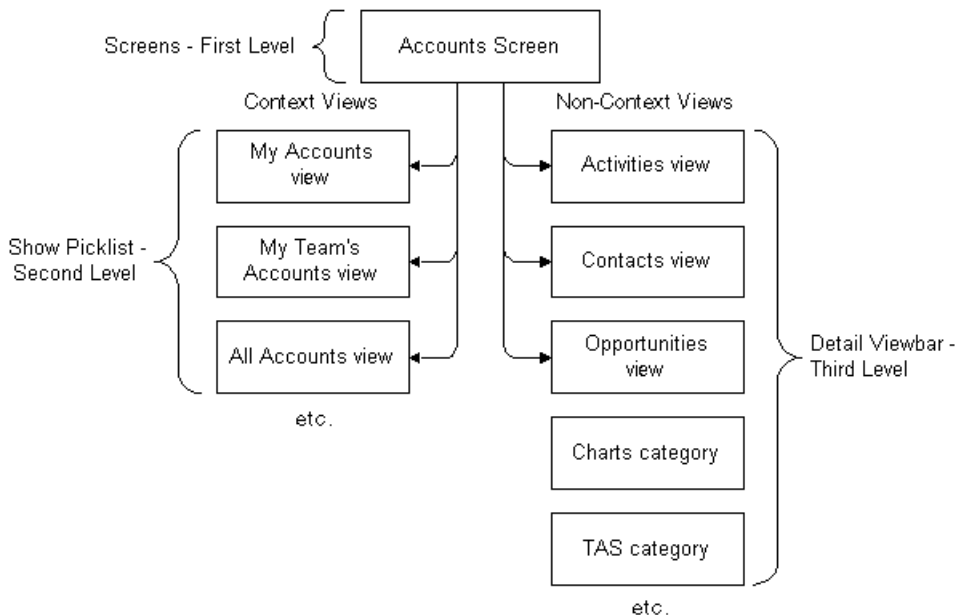


Figure 221. Context and Non-Context Views

Notice that View categories appear with non-context Views in the detail View bar. View categories are described in [“Subcategory Views” on page 809](#).

Context views are automatically segregated from non-context views based on the Business Object and Visibility Applet/Visibility Applet Type properties in each View object definition. The only configuration in Tools required to implement second- and third- level View navigation in a Screen is the assignment of views to that screen using Screen View object definitions. You should avoid using views from different business objects in the same Screen.

The Visibility picklist appears in the “view bar” frame (see the `CCPageContainer.swt` and `CCFrameViewbar.swt` templates). The Visibility picklist is implemented as a `<swe:viewbar>` tag with a `Type` setting of `Select` and a `Mode` setting of `Context`:

```
<swe:form>
  <td nowrap>
    <swe:viewbar type="Select" mode="Context">
      <swe:this property="FormattedHtml"/>
    </swe:viewbar>
  </td>
</swe:form>
```

The detail View bar is also implemented by means of a `<swe:viewbar>` tag, but with different attribute settings. Specifically, the `Type` attribute is omitted, and the `Mode` attribute has a value of `NonContext` instead of `Context`. This creates a horizontal View bar consisting of tabs populated with the display names of all the non-context Views instead of a picklist control populated with the display names of the context Views. The template logic for rendering the detail View bar is as follows (see `CCViewbar_Tabs.swt`):

```
<swe:viewbar mode="NonContext">
  <swe:viewlink state="Active" property="FormattedHtml" >
    <td></td>
    <td class='tier3OnLabel' background="images/nav/tabon_back.gif">
      <noabr>&nbsp; <swe:viewname/> &nbsp;</noabr></td>
```

```

<td></td>

</swe:viewlink>

<swe:viewlink state="Inactive" property="FormattedHtml" >

<td class='tier3Off'><nobr>&nbsp;<swe:viewname/>&nbsp;</nobr></td>

</swe:viewlink>

</swe:viewbar>

```

The detail View bar implementation of the `<swe:viewbar>` tag requires the use of the child tags `<swe:viewlink>` and `<swe:viewname>`. The Visibility picklist implementation omits these child tags.

The syntax of the `<swe:viewbar>` tag appears below:

■ `<swe:viewbar>`

Usage: `<swe:viewbar type="xxx" mode="yyy" property="zzz">`

Attributes:

- **Type.** This can have one value which is "Select." If the type is set to "Select," the view bar is rendered as a HTML select control showing the set of available views (context, non-context or both, depending on the Mode setting). The user navigates to the selected view after making this choice of control.
- **Mode.** The mode can have two values: Context and NonContext. If the value is Context only the context based views will be shown. If the value is NonContext, only the non-context views are shown.
- **Property.** This attribute should be used only when the type is set to Select. This attribute can have a value of FormattedHtml, in which case the HTML select control is rendered.

- `<swe:viewlink>`

Purpose:

Outputs a link to navigate to the view.

Attributes:

- **State.** This is optional. Can have values “Active” or “Inactive.” If state is “Active,” this tag will be used only if the current view name being rendered is the currently active view. If state is “Inactive,” this tag will be used only if the current view name being rendered is not the currently active view. If not specified, the tag will be shown for all views.
- **Property.** This is optional. Can have only one value, “FormattedHtml,” which will output the HTML for creating a link to navigate to the view. If this attribute is not specified, then no output will be generated.
- **htmlAttr.** This is optional. Can be used to add additional HTML attributes to the generated HTML tag.

NOTE: The `swe:viewlink` tag can be used without specifying the property attribute, but with a value for the state attribute to show conditionally different HTML for active and inactive views. When the property attribute is not specified, the property can be displayed within the body of the `swe:viewlink` tag using the `swe:this` tag.

- `<swe:viewname>`

Purpose: Outputs the name of the view.

Subcategory Views

The fourth navigational level is the Subcategory View picklist inside the detail applet, just beneath the row of third-level view tabs. It allows the user to choose a View from the category represented by the selected detail View bar tab, when the View bar tab is for a category rather than an individual View. This is shown in [Figure 222 on page 810](#).

NOTE: The Applet Toggle picklist, which, if present, is inside the applet just beneath the Subcategory View picklist, is not the same as the Subcategory View picklist. The Subcategory View picklist is for View navigation; the Applet Toggle picklist is for selection of an applet within the detail portion of the view. For more information on applet toggles, refer to [“Toggle Applets” on page 849](#).

Categories are groupings of Views within a Screen. If Views are grouped into categories within a Screen, the category names for these Views are rendered in the detail View bar instead of the individual Views, requiring the user to drill down through the subcategory picklist to access the categorized Views. Uncategorized Views appear directly in the detail View bar. For example, in the Accounts screen there is a set of views under the Charts category (including Account Analysis By Rep, State Analysis and Territory Analysis). A detail tab (third navigational level) would be defined for Charts, and when the user clicked on this tab, the Current View picklist (fourth navigational level) would be populated with the names of these three views.

The hierarchy of screens, categories, and categorized views (first level, third level, and fourth level, in this case) is illustrated in [Figure 222](#).

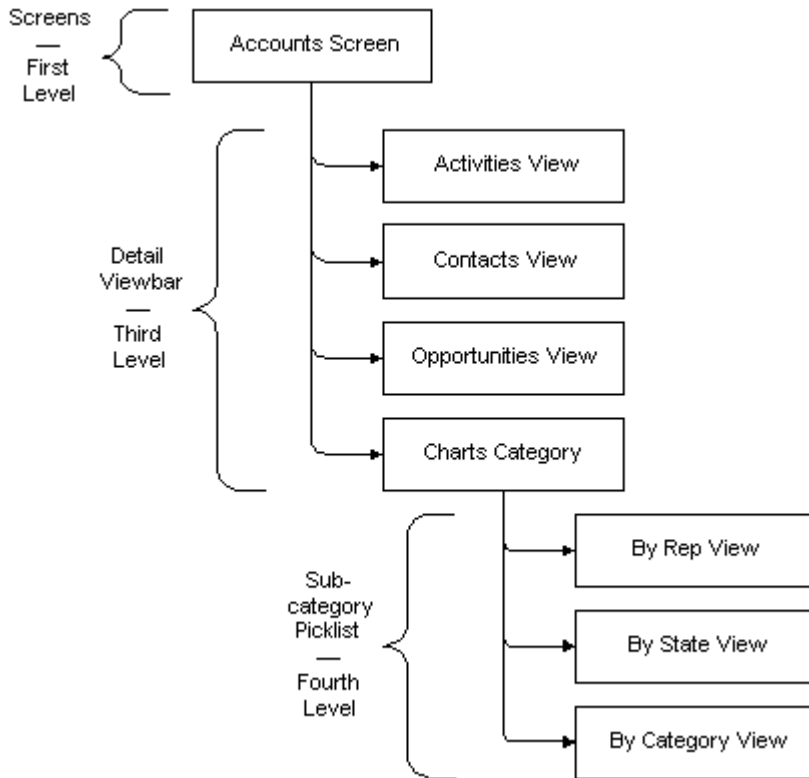


Figure 222. Hierarchy of Levels

Any of the non-context Views in a Screen can be categorized (categories are ignored if specified for context Views). Specification of a category involves the use of the Category property in the Screen View object definition for each View to be included in a category. For example, the Account Analysis By Rep, State Analysis, and Territory Analysis views in the Accounts screen are given a Category value of “Charts.” The Category value needs to be identical in spelling and capitalization among the Views being categorized together in the Screen, or multiple (slightly different) categories will result.

Screen View object definitions with a non-blank Category property result in their corresponding View names being accessible only from the subcategory picklist. A View bar tab is rendered for the category name instead of the names of the individual Views.

A subcategory View picklist is implemented using a `<swe:subviewbar>` tag. The `<swe:subviewbar>` tag can alternately be configured to display a second horizontal tab bar beneath the detail View bar, but the picklist is the preferable approach. It can either be a dropdown picklist or a set of tabs, depending on attribute settings in the tag. The actual placement of it depends where the `<swe:subviewbar>` tag is placed in the template.

The default behavior of the `<swe:viewbar>` tag, if the `<swe:subviewbar>` tag is not present, is to display the default View for that category when the user selects the category name in the detail View bar. The default View is the View with the lowest sequence number in that category that is visible to the user.

By including the `<swe:subviewbar>` tag, this behavior is augmented with a picklist or subcategory tab bar that conditionally appears when the currently active View belongs to a category. This tag expands to list all the Views that belong to the selected category. If the currently active View does not belong to a category, then this tag does not render anything on the page. Thus, if the user chooses a category name from the detail View bar (which means the default View within that category is now the active View), the subcategory picklist or tab bar is rendered in that default View if this tag is present. If the user chooses a non-category View from the detail View bar, the subview bar tag does not render anything on the resulting page.

The `<swe:subviewbar>` tag usage is described as follows:

■ `<swe:subviewbar>`

Usage: `<swe:subviewbar type="xxx" property="zzz">`

Attributes:

- **Type.** This can have one value: "Select". If the type is set to Select, the subview bar will be rendered as a HTML select control (picklist) showing the set of available Views in the selected category.


```
<td></td>

<td class='tier4OnLabel' background="images/nav/tabon_back.gif">
<nobr>&nbsp;<swe:viewname/>&nbsp;</nobr></td>

<td></td>

</swe:viewlink>

<swe:viewlink state="Inactive" property="FormattedHtml" >

<td class='tier4Off'><nobr>&nbsp;<swe:viewname/>&nbsp;</nobr></td>

</swe:viewlink>

</swe:subviewbar>

<td width="100%" class="tier4Back">&nbsp;</td>
```


This chapter discusses these topics:

- [“Page Templates” on page 815](#)
- [“View Templates” on page 825](#)
- [“HTML Frames” on page 817](#)
- [“Applet Templates” on page 828](#)
- [“Toolbars and Menus” on page 865](#)
- [“Thread Bar” on page 871](#)

Page Templates

The application is associated with a set of templates through properties in the Application object definition. These properties include Container Web Page, Error Web Page, Login Web Page, Logoff Acknowledgement Web Page, Sort Web Page, and Acknowledgement Web Page. Each property that is applicable identifies a template to use in the corresponding circumstance.

Acknowledgement Web Page. The Web page displayed after the user logs in. This page is used as the first page the user is taken to after a successful login, except in the case of a login after a time-out. In the case of a login after a time-out, the user is taken to the view to which he or she was trying to navigate when the time-out occurred.

Acknowledgement Web View. The Siebel view displayed after login. This page is used as the first view the user is taken to after a successful login, except when:

- A user logs in after a time-out. In this case, the user is taken to the view to which he or she was trying to navigate when the time-out occurred.

- Explicit login is specified for an SI mode view. In a standard interactive application such as eService, if view accessible through the home page as the Explicit Login property is set to TRUE (anonymous browsing), then after successfully entering the login credentials, the user is taken to this view instead of the Acknowledgement Web View defined in this property.

Container Web Page. A page that defines the structure of the application. This page can contain the common UI components like screen bars, view bars, logos, and so on. This page can be used to define the HTML Frame definition document for the application. All views and pages (optionally) are shown within the context of the container page. [“Web Page-Layout Container Page” on page 816.](#)

Error Web Page. The page to use when an error occurs in the application.

Login Web Page. The page to use as the Login page.

Logoff Acknowledgement Web Page. The page to which the user is taken to after logging off the application.

Sort Web Page. The page to be used to create a dialog to perform an advanced sort of list applet columns.

Web Page-Layout Container Page

The Web Page-Layout (Container Page) contains markup language and SWE tag elements that define the Web equivalent of the application window. You can see this template's logic in CCPageContainer.swt. The container page template, like view and applet templates, is processed by the Siebel Web Engine.

The container page is the outermost template; it references view templates that in turn reference applet templates. For further information on the Web Page Layout (Container Page), see [Chapter 4, “Application Configuration \(Basic Concepts\).”](#)

Container Page Areas

In the Web Page Layout Container Page, you find the following elements:

- The top of the container page contains markup such as corporate banner, as well as Siebel tags for predefined queries (favorites).

- The screen (tab) bar is generated beneath these as a table, and loaded by means of the SWE logic associated with the `<swe:screenbar>` and `<swe:screenlink>` tags.
- The view bar is also loaded, into the left-hand portion of the page, by means of the `<swe:viewbar>` and `<swe:viewlink>` tags.

Once the container page is loaded, with screen and view names displayed, the screen and view names function as hyperlinks.

- When a screen name is clicked, the template for the default view for that screen is obtained, and the view is generated and displayed.
- When a view name is clicked in the view bar, the view template that is referenced in the view's object definition is loaded.

The Siebel Web Engine processes the set of tags in the view template to incorporate applets into the page. The view object definition identifies the applets to appear in specific sectors, and the templates for these are obtained. Similarly, tag references to controls in each applet are resolved by obtaining the corresponding controls from the repository, which are loaded into the Web page as specified in the applet's template. The container page can contain frames to support independent updating and scrolling of the various areas of a page. The use of frames is described in the next section.

HTML Frames

HTML frames are available to use in the application's container page and in View templates.

Physical UI Navigation and Templates

Page Templates

Frames are used in the container page of the application to provide independent updating and scrolling of each of the three navigation and control areas—toolbars, tab bar, and application menus/View picklist—as well as the content (View) area. This is illustrated in [Figure 223](#).

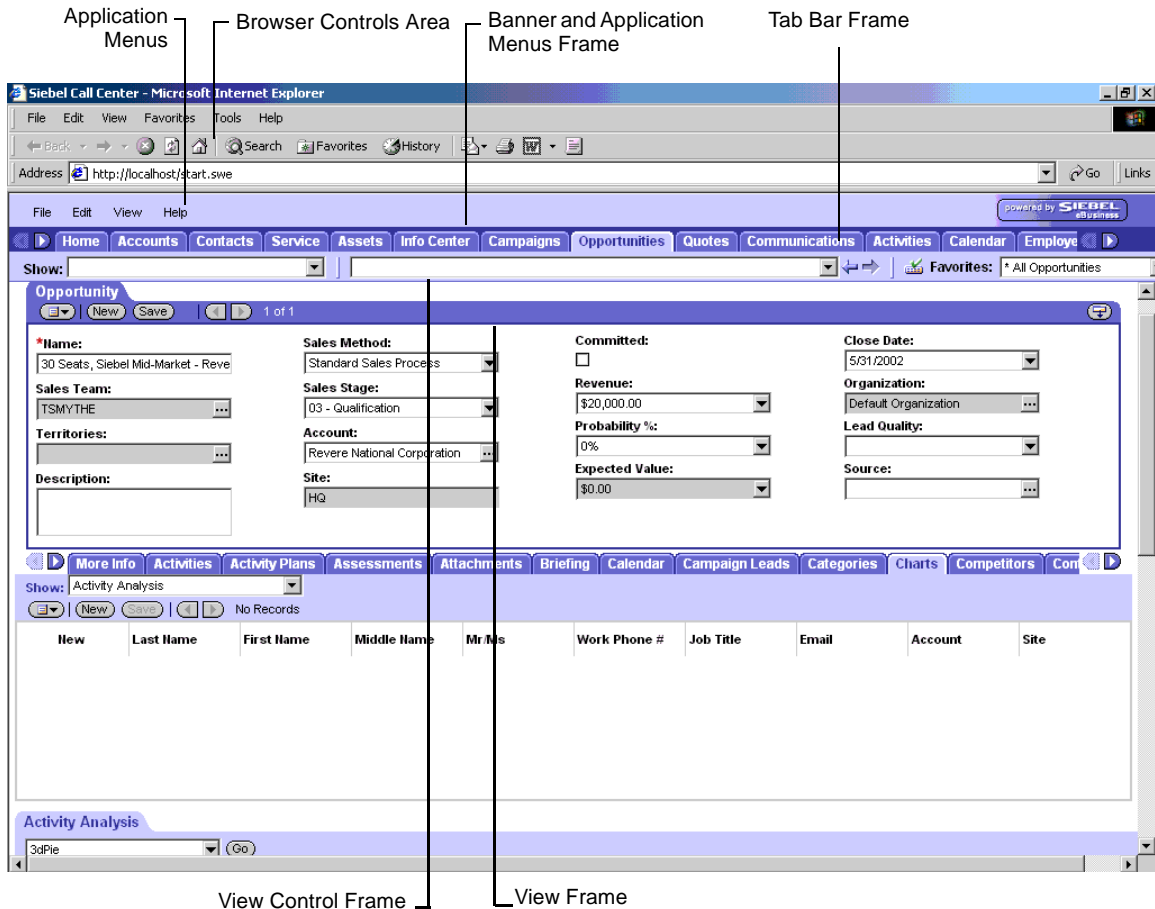


Figure 223. HTML Frame

In a View template, applets can be grouped into separate frames, although this is considered a non-standard practice except in cases where independent refresh or independent scrolling is a significant requirement.

Rather than using the HTML `<frame>` and `<frameset>` tags, the `<swe:frame>` and `<swe:frameset>` tags are used in the Siebel applications so that SWE is aware of the frame names, and can control refresh and the targeting of URLs. These two SWE tags are described as follows:

■ `<swe:frameset>`

Purpose: This tag is analogous to the HTML frameset tag and is used to define the set of frames contained in the document. This tag is rendered by SWE as an HTML `<frameset>` tag. The body of this tag can only contain the `<swe:frame>` tags described below.

Usage: `<swe:frameset htmlAttr="xxx"> ... </swe:frameset>`

Attributes:

- **htmlAttr.** This attribute can be used to specify the attributes for the HTML `<frameset>` tag. For example, `htmlAttr="rows='89,25,*'"` will support a layout in which the frames that belong to the frameset will take up 89 pixels, 29 pixels, and the rest of the window respectively.

■ `<swe:frame>`

Purpose: This tag is used to mark the beginning and end of the contents to be placed into a frame. SWE renders this tag as an HTML `<frame>` tag, with its `src` attribute set to a SWE URL that will retrieve the contents of the frame. This tag should be placed within the body of the `<swe:frameset>` tag.

Usage: `<swe:frame type="xxx" name="yyy"> ... </swe:frame>`

Attributes:

- **Type.** The type attribute is used to indicate the nature of the contents of the frame. SWE uses this information to decide when to refresh this frame. SWE supports the following values for this attribute.
 - **Toolbar.** In a container page template, specifies that the frame contains the tool bar.

- ❑ **Screenbar.** In a container page template, specifies that the frame contains the primary tab bar.
- ❑ **Viewbar.** In a container page template, specifies that the frame contains the application menus, Visibility picklist, and Search picklist.
- ❑ **View.** In a container page template, specifies that the frame contains the current view, that is, the content area.
- ❑ **Page.** In a container page template, specifies that the frame contains a Web page. These frames will not be refreshed after initially loading.
- ❑ **Applet.** In a View template, specifies that the frame contains an applet.
- ❑ **Content.** Supports multiple views on a page. The Content type frame defines the content area. It will contain a frame of type View that shows the main view. It can also contain one or more frames of type AltView to show alternate views, like the search center.
- ❑ **AltView.** Used to designate subframes to show one or more alternate views in the content frame, such as the search center, in addition to the one in the View frame.
- ❑ **Name.** This attribute can be used only when the type of the frame is Page. In this case, you can use this attribute to specify a name for the frame. For other frame types, SWE will generate standard names for the frames.

NOTE: SWE supports nested framesets. In this case the `<swe:frame>` tag will contain a `<swe:frameset>` tag, and the Type attribute of the outer `<swe:frame>` tag is set to Page.

HTML Frames in Container Page Templates

A container page template is used to create the frame definition document for the application. Note the following implementation details of `<swe:frame>` and `<swe:frameset>` tags in container pages:

- You do not have to define the contents of a frame using the `<swe:include>` tag although it is recommended. The contents can be placed directly into the body of the `<swe:frame>` tag.

- The contents of the `<swe:frame>` have to be complete HTML documents, that is, they should contain the HTML document structure tags like `<html>`, `<head>`, `<body>`, and so on. This includes the view templates as well.
- The contents of the `<swe:frame>` tag when the type is View should contain only the `<swe:current-view/>` tag.

The following `<swe:frameset>` definition is from the standard container page, `CCPageContainer.swt`:

```
<swe:frameset htmlAttr="rows='60,21,25,*' border='0'
frameborder='No'">

  <swe:frame type="Page" htmlAttr="marginheight='0'
marginwidth='0'noresize scrolling='No'">

    <swe:include file="CCFrameBanner.swt"/>

  </swe:frame>

  <swe:frame type="Screenbar" htmlAttr="marginheight='0'
marginwidth='0' noresize scrolling='No'">

    <swe:include file="CCFrameScreenbar.swt"/>

  </swe:frame>

  <swe:frame type="Viewbar" htmlAttr="marginheight='0'
marginwidth='0' noresize scrolling='No'">

    <swe:include file="CCFrameViewbar.swt"/>

  </swe:frame>

  <swe:frame type="View" htmlAttr="marginheight='0'
marginwidth='0' noresize scrolling='Auto'">

    <swe:current-view/>

  </swe:frame>

</swe:frameset>
```

Support for Multiple Views on a Page

The SWE framework supports showing multiple views simultaneously on a page. The multiple views consist of a Main view and one or more Alternate views. The main view is the view that is selected using the view bar (level two or three) for a given screen. There is only one main view. Alternate views are other views that can be shown along with the main view: for example, the Search View that shows applets that can be used for find/search operations.

The multiple views shown on a page can be placed into separate HTML frames or can share the same frame. Multiple views can also be shown with the main view in the main browser window and a single alternate view in a pop-up window.

In employee applications that use high interactivity, only the main view can be in high interactivity. The alternate views will be shown in standard interactivity. It is recommended that you configure alternate views as simple views without any complex navigation links.

NOTE: The examples given here describe creating multiple view layouts when HTML frames are used. The process is similar when frames are not used. In such cases, HTML tables can be used in the place of frames and framesets to position the views.

To support multiple views, the structure of framesets and frames used in the application has to be modified. Defined framesets and frames in the application's container template and in the view template were discussed earlier in chapter. In addition, there is another layer, the Content Container (the container page for the Content area).

The frame of type View which was in the Application's Container page should be replaced with a frame of type Content. This frame defines the area where one or more views can be loaded. Initially this frame will contain a frameset that will have the View type frame.

The structure of the container template is given in the example below:

```
<swe:frameset htmlAttr="rows='80,50,50,*' border='0'
frameborder='No' ">
```

```
    <swe:frame type="Page" htmlAttr="marginheight='0'
marginwidth='0' noresize scrolling='No' ">
```

```
<swe:include file="CCBanner.swt"/>

</swe:frame>

<swe:frame type="Screenbar" htmlAttr="marginheight='0'
marginwidth='0' noresize scrolling='No'">

    <swe:include file="CCScreenbar.swt"/>

</swe:frame>

<swe:frame type="Viewbar" htmlAttr="marginheight='0'
marginwidth='0' noresize scrolling='No'">

    <swe:include file="CCViewbar.swt"/>

</swe:frame>

<swe:frame type="Content" htmlAttr="marginheight='0'
marginwidth='0' noresize scrolling='Yes'">

    <swe:include file="CCMainView.swt"/>

</swe:frame>

</swe:frameset>
```

The file `CCMainView.swt` defines a frameset that contains the main view.

```
<swe:frameset htmlAttr="cols='100%' border='0' frameborder='No'">

    <swe:frame type="View" htmlAttr=" noresize scrolling='Yes'">

        <swe:current-view/>

    </swe:frame>

</swe:frameset>
```

After making this change, the application should behave as before. All that was changed was the introduction of one additional layering of frames in the content area. The previous application container page template that had the View frame without the outer Content frame does not generate any errors, but does not allow showing multiple views in the application. All the application container templates should be modified to use the Content frame.

To show additional views in the content area, a different Content Container page in the Content frame should be loaded. This can be done by invoking the method `LoadContentContainer` from a control or page item. The Content Container to be loaded should be passed in using the User Property Container.

NOTE: This should be set to the Web Template Name of the content container page and not to the .SWT file name. For example, to show the search view along with the main view, create a content container page (for example, `CCSMainAndSearchView.swt`), and load it using the `LoadContentContainer` method. `CCSMainAndSearchView.swt` contains the tags to load the main view and search view into two frames as shown:

```
<swe:frameset htmlAttr="cols='100%' border='0' frameborder='No'">
  <swe:frame type="View" htmlAttr="noresize scrolling='Yes'">
    <swe:current-view/>
  </swe:frame>
  <swe:frame type="AltView" name="Search" htmlAttr="noresize
  scrolling='Yes'">
    <swe:view name="Search View" id="Search" />
  </swe:frame>
</swe:frameset>
```

The main view is still called the `<swe:current-view>` tag. Alternate views are referred to using the `<swe:view>` tag.

■ `<swe:view>`

Syntax:

```
<swe:view name="xxx" id="yyy">
```

Attributes:

■ **Name.** Name of the Alternate View

- **Id.** An Id for the location (or zone) occupied by this view. This Id will be used to replace this view with another view in its place.

The `<swe:frame>` tag contains alternate views called AltView.

To switch from showing the Search and Main views to showing only the Main View, invoke the `LoadContentContainer` method again, this time passing in the `CCMainView.swt` based container page.

View Templates

A view is a collection of applets displayed on a screen at the same time. It consists of a single window displaying related data forms and lists (applets). The user can select the current (active) view from either the Screenbar (the default view for that screen), the second-level Visibility picklist, a third-level tab, the fourth level Category-view picklist, the thread bar, the history list, history forward and back buttons, or by a drilldown from another view. Access to particular views is determined by the four navigational constructs in the physical UI.

View Templates are associated with a view through the View Web Template object definition. A view template primarily contains placeholders for applets as specified by the `<swe:applet>` tag. The mapping of specific applets to these placeholders is done visually through the View Web layout editor. The following is an example of a view template.

```
<!-- Template Start: CCViewBasic.swt -->
<!------- Page Title ----->
<title>
<swe:this property="Title"/>
</title>
<!------- Salutation applet and Search Applet, table 3.1 ----->
<table border="0" cellspacing="0" cellpadding="1" width="100%">
  <tr>
    <td width="66%"><swe:applet id="101"/>&nbsp;  </td>
```

```
<td width="33%"><swe:applet id="201"/>&nbsp;</td>
</tr>
</table>
<!-- End Salutation applet and Search Applet, table 3.1 -->
<!-- Regular Applet(s) ---->
<swe:for-each count=5 iteratorName="currentId" startValue="1">
  <swe:applet id="swe:currentId"/>
</swe:for-each>
<!-- Special Applet(s) ---->
<swe:for-each count=3 iteratorName="currentId" startValue="11">
  <swe:applet id="swe:currentId"/>
</swe:for-each>
<!-- Template End: CCViewBasic.swt -->
```

Notice that each `<swe:applet id=x>` tag acts as a placeholder for an applet's location in the view template. This same view template can be used to show different views by mapping the view's applets to these placeholders. In the default view templates shipped with Siebel Applications, `swe:applet` tags with IDs of 101 and 201 are used to show the salutation and search applets at the top of the views. The IDs 1 through 10 are used to show the main applets in the view, and the IDs starting with 11 are used to show some special applets that appear at the bottom of some views.

HTML Frames in View Templates

HTML frames can be used in View templates to create a frame definition document to show the Applets in the View. SWE will refresh these frames only when one or more of the Applets contained in a frame has new data.

NOTE: You can use frames in a View template only if frames are also used in the container page and there is a separate frame in the container page for the View.

In a View template, applets can be grouped into separate frames, although this is considered a non-standard practice except in cases where independent refresh or independent scrolling is a significant requirement. One situation where frames are required in the content area is when displaying an explorer View, in which a tree applet occupies a frame on the left and the corresponding list applet occupies the frame on the right. Another situation requiring frames is when the user activates a search, at which time a Search frame, and subsequently a Results frame, are activated in the right portion of the content area.

The following shows the implementation details of frameset definitions in View templates:

- When placing Applets into frames you need to make sure that at least one `<swe:applet>` tag within a frame gets mapped to an Applet in the repository. Otherwise empty frames will occur.
- When a `<swe:frame>` block contains a `<swe:applet>` tag, its type attribute should be set to Applet.

Given below is an example of a view template that uses frames:

```
<!-- CCView_33_66_Frame.swt start -->

<swe:frameset htmlAttr="cols='33%,66%'" border='1'
frameborder='Yes' ">

<!-- Column 1 Applets -->

<swe:frame type="Applet" htmlAttr="marginheight='0' margin
width='0' scrolling='Auto'">

<swe:for-each count=10 iteratorName="currentId" startValue="101">

    <swe:applet id="swe:currentId" hintText="Applet" var="Parent">

        <!--start applet-->

            <swe:this property="FormattedHtml"/>

        <!--end applet-->

    </swe:applet>

</swe:for-each>
```

```
</swe:frame>

<!-- Column 2 Applets -->

<swe:frame type="Applet" htmlAttr="marginheight='0' marginwidth='0'
scrolling='Auto' ">

<swe:for-each count=10 iteratorName="currentId" startValue="201">

    <swe:applet id="swe:currentId" hintText="Applet" var="Parent">

        <!--start applet-->

            <swe:this property="FormattedHtml" />

        <!--end applet-->

    </swe:applet>

</swe:for-each>

</swe:frame>

</swe:frameset>

<!-- CCView_33_66_Frame.swt end --> </HTML>
```

Applet Templates

The Applet Web Template child object type (of Applet) makes it possible to specify multiple templates for a single applet, each template file associated with one or more modes. The Applet Web Template object type has the following important properties:

- **Name.** Indicates the edit mode that the applet template supports (such as Edit or New).
- **Web Template.** Provides the name of the Web Template used for that mode.

The Applet Web Template Item child object type (of Applet Web Template) defines the mappings between controls and list columns to placeholders in the Web template file. The Applet Web Template Item object type has the following properties:

- **Name.** Name of the object definition, generally the same as the Control property.
- **Control.** Specifies the name of the control as it is to appear.
- **Item Identifier.** This specifies a unique numeric identifier for each control, generated in the layout editor. The value is used in the markup language tag that specifies the corresponding control in a template, binding the control to a specific position on the page.
- **Type.** Consists of the value Control, List Item, or Web Control, indicating what kind of control the applet Web template item defines.

The following types of applets are discussed in this section:

- Form applets
- List applets (including MVG and pick applets)
- Toggle applets
- Tree applets
- Chart applets
- Pop-up applets
- Catalog-style list applets and rich lists

Form Applets

A Form Applet can appear in any of the four major modes—Base, Edit, New, and Query. The following is an example of a Form Applet template for use in Edit, New, and Query modes. Applets to be used in Base mode are similar except that they do not contain the `<swe:form>` tag.

```
<swe:control id="1100">
  <div class=CmdTxt>
    <swe:this property="FormattedHtml" hintText="Outside Applet
      Help Text"/>
  </div>
</swe:control>
```

```
<table class="AppletStyle1" width="100%" align="center">
  <swe:form>
    <tr>
      <td colspan="2">
        <swe:include file="CCTitle.swt"/>
      </td>
    </tr>
    <tr>
      <td>
        <swe:error>
          <swe:this property="FormattedHtml"/>
        </swe:error>
      </td>
    </tr>
    <swe:for-each startValue="1301" count="10"
      iteratorName="currentId">
      <swe:control id="swe:currentId" hintMapType="FormItem">
        <tr valign="top">
          <td class="scLabelRight">&nbsp;
            <swe:this property="RequiredIndicator"
              hintText="Required"/>
            <swe:this property="DisplayName" hintText="Label"/>
          </td>
          <td class="scField">
            <swe:this property="FormattedHtml" hintText="Field"/>&nbsp;
          </td>
        </tr>
      </swe:control>
    </swe:for-each>
  </swe:form>
</table>
```

```
        </tr>
    </swe:control>
</swe:for-each>
</swe:form>
</table>
```

The main tags that appear in this template are `<swe:form>`, `<swe:control>`, and `<swe:error>`.

■ `<swe:form>`

The `<swe:form>` tag is analogous to an HTML `<form>` tag and encloses a section of a page that accepts user input. The main attributes of this tag are `htmlAttr` and `Name`, both of which are optional.

The values of the `htmlAttr` should be valid attributes of the HTML `<form>` tag other than `method`, `name`, and `action`. These attributes will be used as is with the HTML `<form>` tag that is generated. The `name` attribute creates an HTML form with the specified name. If this attribute is not specified, an internally generated name is used.

■ `<swe:control>`

The `<swe:control>` tag specifies placeholders for controls. The main attributes of this tag are `id`, which maps the control to the placeholder, and the `property`, which specifies the property of the control to be rendered. The values for the `property` attribute that are germane to form applets include `FormattedHTML`, `DisplayName`, and `RequiredIndicator`.

The `FormattedHTML` property causes the data value of the control to be rendered, while the `DisplayName` corresponds to the `Caption` property. The `RequiredIndicator` results in specific HTML being rendered if the underlying Business Component Field is required.

■ <swe:error>

When a server side error occurs on submitting a form, the same page will be shown again with the error message displayed within the page. The <swe:error> tag denotes the location of this error message. The only attribute of the tag is a property whose value must be FormattedHtml. This results in the contents of the error message to be displayed. If when the form is rendered there are no errors, the contents of the <swe:error> tag are skipped.

NOTE: For errors that occur outside of a form submission, the application's Error Page will be used.

In applications where the task activities are primarily data editing and input, you can forego read-only forms (Base mode) and use persistently editable forms. This type of form saves considerable time since data can be entered without first clicking an edit button and waiting for the form to appear in edit mode. An illustration of a persistently editable form applet appears in [Figure 224](#).

Editable Text Field

*Name: 30 Seats, Siebel Mid-Market - Reve	Sales Method: Standard Sales Process	Committed: <input type="checkbox"/>	*Close Date: 5/31/2002
Sales Team: TSMYTHE	Sales Stage: 03 - Qualification	Revenue: \$20,000.00	Organization: Default Organization
Territories: [empty]	Account: Revere National Corporation	Probability %: 45%	Lead Quality: 2-Very High
Description: 30 Inside Sales Professionals - really blended agents who...	Site: HQ	Expected Value: \$9,000.00	Source: [empty]

Figure 224. Editable Form Applet

If an applet is set to be in the Edit mode in a view (as specified by the mode property of the View Web Template Item), this applet is never shown in the Base mode. If you update the field values in this applet and commit the change, the applet continues to be shown in this mode after the changes are written to the database. You can, however, invoke a method like NewQuery or NewRecord on an applet that is shown in an Edit mode to show it in the Query or New modes. After executing the query or writing the new record, the applet is shown in the Edit mode.

To show a form applet in Edit mode in a view

- 1** Update the control used to save the record so that it invokes the ExplicitWriteRecord method.

In a persistently editable form applet, use the ExplicitWriteRecord method (HTML Row Sensitive property set to TRUE) instead of WriteRecord.

- 2** Map this control to the Applet Web Template of type Edit.

For an applet that you are showing in the Edit mode in a view, you do not need to specify a Base type template, as the applet is never shown in the Base mode.

Because the applet is being shown in the Edit mode by default in the view, you do not need the EditRecord, WriteRecord, or UndoRecord controls. You only need the ExplicitWriteRecord and ResetRecord controls.

- 3** Map the applet to the view template in the Edit mode.

In Siebel Tools, the View Web Template Item object type has a property called Applet Mode. This property is used to specify the mode to be used for the applet when rendering the view. The default value is Base. To make the form applet editable in the view, set the value of the Applet Mode property to Edit when mapping the applet in the View Web Template.

List Applets

Standard list applets allow simultaneous display of data from multiple records. The standard list displays data fields in a multi-column layout with each record of data represented in a row. In addition to textual data, lists also support images in JPEG and GIF formats and edit controls such as check boxes, drop-down lists, MVGs, and text fields.

A single row at a time within the standard list applet is selected for editing by clicking in the far left column selection area. When selected, the fields within the row can activate either input or edit controls. Clicking the New button creates a new row with a series of blank fields for the user to populate. A standard list applet in edit mode is shown in [Figure 225](#).

Account:

New	Name	Site	Main Phone#	Territories	Industries	Status	URL
	Woolen Goodrick N	Boston	(617) 244-5020		animal specialties	Active	
	3Com	Santa Clara	(408) 326-5000		manufacturing indus	Gold	www.3com.com
	AMCO Communicati	Chicago, IL	(847) 491-2300		steel pipe & tubes	Active	www.amco.net
*	Alliance Program						
	Andercotti Consultin	Palo Alto	(415) 389-5400		services	Active	
	Andronico	Berkeley	(510) 568-1200		groceries & related	Active	

Figure 225. Standard List Applet

Persistently Editable List Applets

Edit List applets are shown in a persistently editable mode in views that use high interactivity. The purpose of an editable list applet is to allow users to modify the records in a list applet without switching to an edit page.

The editable list applet has the following features:

- Editable cells displayed as text input, list box, or mini buttons
- Modified records that can be saved individually

Edit List mode renders list applets persistently editable. A list applet that is rendered in this mode can still be shown in the Edit mode by invoking the EditRecord method. Typically, the Edit List mode is used for editing the most commonly updated fields, and the Edit mode is used to edit the record.

A given list applet exhibits different behavior and appears differently depending on whether it is part of a view being displayed using standard or high interactivity:

- In standard interactivity, there is a row selector control that allows the user to select a specific row for editing. When selected, the fields within the row can activate either input or edit controls. Clicking the New button creates a new row with a series of blank fields for the user to populate.
- Row selection using high interactivity is done by clicking on any area within a row in the list. Therefore, the row selection is redundant, and the control is automatically deleted when the list is rendered. Also, because high interactivity uses an implicit save model, a Save control is not required. When a user steps off the current record, the changes are automatically saved. A list applet in Edit List mode is shown in [Figure 225 on page 834](#).

To show list applets in Edit List mode

- 1** Add new controls for the applet that are required to support Editable Lists.

The editable list applets will be rendered in a view with the currently selected row showing editable fields as data entry (as opposed to read-only) controls. Users can change the values of these fields and then use a Save control to commit the changes to the database. To update another row, the user has to select that row, upon which the fields in the previously selected row return to read-only status and the fields in the newly selected row change to data entry controls. To obtain this behavior, the following two controls should be added to the list applet.

- **Control to Select a Row.** This control should invoke the method `PositionOnRow`, and its HTML Row Sensitive property should be set to `TRUE`. This control appears once per row. (Refer to [“Current Record Selection in List Applets” on page 840](#) for more details.)
- **Control to Save a Row.** This control should invoke the method `WriteRecord`, and its HTML Row Sensitive property should be set to `TRUE`. This should appear once for the entire list.

- 2 Mark the list columns that you want to edit in the list applet.

By default, all list columns in the list applet are editable in the Edit List mode if the business component allows updates to the field and if the Read Only property of the list column is not set to FALSE. However, having all columns editable is not recommended. Only the fields that would require frequent updates should be shown as editable. You still must provide an Edit Record link to edit all the fields in the record. To mark which columns in the list applet should not be editable when rendering the applet as an editable list, set the property HTML List Edit of the list column to FALSE (The default is TRUE).

- 3 Set the template to be used for rendering the applet in the Edit List mode.

In Tools, add a new Applet Web Template for the list applet, where the Type property is set to Edit List, and the Web Template is set to the template to be used for rendering the applet in this mode. Add the two controls mentioned above as Applet Web Template Items for this template.

NOTE: For an applet showing the Edit List mode, you do not need to specify a Base type template since the applet is never shown in the base mode. Also, you can share the same Web Template object between list applets in the Base and Edit List modes.

- 4 Map the list applet to the view template in the Edit List mode.

The View Web Template Item has a property called Applet Mode, which is used to specify the mode to be used for the applet when rendering the view. The default value is Base. To make the list applet editable in the view, set the value of this property to Edit List when mapping the applet in the View Web Template.

Sample List Applet Template

```
<table width="100%" cellspacing="0" cellpadding="0" border="0"
align="center">

<swe:form>

    ...

<swe:list>

<!-- List Header Section Start>
```



```
<swe:control id="147">
  <td class="Header" align="center">
    <swe:this property="DisplayName"/>
  </td>
</swe:control>
<swe:select-row>
  <td width="42" align="center" class="Header">&nbsp;
  </td>
</swe:select-row>
<swe:for-each startValue="501" count="20" iteratorName="currentId">
  <swe:control id="swe:currentId">
    <td align="swe:this.TextAlignment" class="Header">
      <swe:this property="ListHeader"/>
    </td>
  </swe:control>
</swe:for-each>
<swe:control id="142">
  <td class="Header" align="center">
    <swe:this property="DisplayName"/>
  </td>
</swe:control>
<!-- List Header Section End>
<!------- Loop for all 7 records, List Body ----->
<swe:for-each-row count="7">
<tr class="swe:this.RowStyle">
```

```
<swe:control id="147">
  <td width="42" align="center" class="Row">
    <swe:this property="FormattedHtml" hintMapType="Control"/>
  </td>
</swe:control>

<swe:select-row>
  <td width="42" align="center" class="Row">
    <swe:this property="FormattedHtml" />
  </td>
</swe:select-row>

<!-- ----- List Field Values (501-520) ----->
<swe:for-each startValue="501" count="40" iteratorName="currentId">
  <swe:control id="swe:currentId">
    <td align="swe:this.TextAlignment" class="Row">
      <swe:this property="FormattedHtml"
        hintText="Field"/>
    </td>
  </swe:control>
</swe:for-each>

<!-- ----- Per-record Control Buttons ----->
<swe:control id="142">
  <td align="center" class="Row">
    <swe:this property="FormattedHtml" hintMapType="Control"/>
  </td>
</swe:control>
```

```
</tr>
</swe:for-each-row>
<!-- ----- End Loop, List Body ----->
</swe:list>
...
</swe:form>
</table>
```

Tags that typically appear in a list applet template include `swe:form`, `swe:list`, `swe:control`, `swe:select-row`, and `swe:for-each-row`.

■ `<swe:form>`

As with form applets, this tag encloses an editable section. It is therefore used for editable list applets.

■ `<swe:list>`

This tag encloses the section of the template that contains the list header and body. For applications that use high interactivity, the section between the start and end of the `swe:list` tags is replaced by the specialized List Control that supports capabilities such as resizing columns, and so on. This tag is ignored for standard interactivity applications.

■ `<swe:control>`

This tag defines a placeholder for List Columns. The property attribute takes the same values as in the case of form applets - `DisplayName` for the Display Name attribute of the list column object, `FormattedHTML` for the data value. In addition, certain attributes of a list column object can be used to control the attributes of an HTML element contained within the enclosing `swe:control` tag. For example, the `align` attribute of a contained TD tag can be set to be equal to the `Text Alignment` property of the enclosing list column as follows:

```
<td align="swe:this.TextAlignment">
```

- `<swe:select-row>`

This tag is used to render check boxes for selecting a row for the purposes of multi-selection. This is described in greater detail in the section on multi-select lists.

- `<swe:for-each-row>`

This tag encloses the section of the template that is to be repeated for each list row.

Current Record Selection in List Applets

The Web client has a feature that allows users to select a record as the currently active record in a list applet in the Base and the Edit List modes.

NOTE: This applies only to standard interactivity applications. For high interactivity applications, row selection is achieved by clicking anywhere within the current row.

To select a record as the currently active record

- 1** Add a control to all list applets that invokes the method `PositionOnRow`.
- 2** The HTML Row Sensitive property of this method should be set to `TRUE`.
- 3** Place this control on the list applet where you want the link to select the row. You are able to select the record by clicking on this link.

SWE provides two options to show the currently selected record. These options can be used together or individually:

- 1** You are able to specify the formatting to be used for the currently selected row.

This feature is based on changing the CSS style sheet class associated with a row (for example in a `<TR>` tag) to specify different formatting information. You can associate a list applet with a named style to be used for formatting its rows. You can define this attribute in the `cfg` file used by the application under the SWE section. (This is currently limited to all list applets used by an application so they have the same row formatting style.)

```
ListRowStyle           = "Siebel List"
```

You can specify any name for the row style. The actual style sheet classes used by this named style is specified in a new file type called the Siebel Web Style file (similar to the Siebel Web Format file used for custom html types). The Siebel Web Style files (SWS files) will have the extension .sws, and should be installed in the same folder as the template files.

As in the case of the SWF files, the SWS files used by an application are specified in the .cfg file of the application. There are two parameters that define the SWS file. One defines the file defined by the Siebel application teams and the other the file that can be defined by customers to either override the styles specified by the application teams or to add new styles. The parameters should be defined in the SWE section of the cfg file.

```
SystemSWSName          = CCStyles.sws  
;UserSWSName           = // for customer use only
```

In the SWS file you can define the style sheet classes to be used with a named style using two new SWE tags and conditional tags like `<swe:if>` or `<swe:switch>`. These tags are described as follows:

- `swe:style`

Usage: `<swe:style type="XXX" name="YYY">`

Attributes:

- **type.** Currently supports only one value which is RowStyle. Other values will be supported in the future.

- **name.** Name of the style (like Siebel List)

swe:class

Usage: <swe:class name="XXX" />

Attributes:

name. Name of the CSS style sheet class. The style sheet that defines this class should be loaded through the template.

The following is a sample entry in a SWS file that will achieve the same formatting as the conditional tags that were put in the template file using the earlier approach:

```
<swe:style type="RowStyle" name="Siebel List">
<swe:switch>
  <swe:case condition="Web Engine State Properties, IsErrorRow">
    <swe:class name="listRowError" />
  </swe:case>
  <swe:case condition="Web Engine State Properties,
IsCurrentRow">
<swe:class name="listRowOn" />
  </swe:case>
  <swe:case condition="Web Engine State Properties,
IsOddRow">
<swe:class name="listRowOdd" />
  </swe:case>
  <swe:case condition="Web Engine State Properties,
IsEvenRow">
<swe:class name="listRowEven" />
  </swe:case>
<swe:default>
```

```
<swe:class name="listRowOff" />

</swe:default>

</swe:switch>

</swe:style>
```

In the template file used by the list applet, the conditional tags used earlier should be replaced with a new property of the applet called “RowStyle” that can be set to the class attribute of any HTML tag. The format used for specifying the RowStyle property of the list applet is similar to that used for specifying the TextAlignment property of a list column.

```
<swe:for-each-row count="7">

<tr class="swe:this.RowStyle">

<swe:for-each startValue="501" count="20"
iteratorName="currentId">

  <swe:control id="swe:currentId">

    <td align="swe:this.TextAlignment" class="Row"><swe:this
property="FormattedHtml" hintText="Field"
hintMapType="ListItem" /></td>

  </swe:control>

</swe:for-each>

</tr>

</swe:for-each-row>
```

- 2** You can use the PositionOnRow control itself to distinguish between the selected and the unselected rows. Once a row is selected, the PositionOnRow control on that row will be in a disabled state. So you can use different images for the disabled and enabled state of the control to differentiate between selected and unselected rows.

Multi-Row Editable List Applets

This is an extension to the Editable List Applet capability. By default when an applet is rendered in the Edit List mode in a view, only the currently selected row is editable. To edit other rows, you need to save the current changes and then select the next row to edit.

It is also possible to render list applets in Edit List mode where all the rows are editable. Users can update multiple rows and then save all the records with one invocation of the SaveEditRecord control.

To make a list applet support multi-row edits in Edit List mode, set the HTML Multi Row Edit property of the List object in tools to TRUE. The default for this attribute is FALSE. All the other steps are the same as for the regular Edit List mode.

NOTE: You do not need to place the SaveEditRecord control on each row. Only one such control is required for the applet.

There are certain limitations around the usage of this feature:

- If an error occurs while committing any of the records, the Siebel Web Engine will try to commit as many of the records that it can and will report errors on all the failed records. However, the error messages may not have sufficient information on which rows failed.
- Changes in the current working set must be committed before you can navigate to another working set (in other words you need to save your changes before calling GotoNextSet, GotoPreviousSet, and so on).

Because of these limitations, this feature should be used only in cases where these limitations will not cause a significant impact on the application's usability.

Hence this feature should be used only when the following conditions are met:

- Validation errors in the editable fields of the applet can be caught with client side validation (using the Browser Script).
- Only one user will be updating the records of this applet at any given time.
- The number of records in the list applet are small enough that they can be rendered on a single page without the Next or Previous controls.

A good example of the use of this feature is to update the Quantity field in the Shopping Cart applet.

NOTE: This feature is specific to standard interactivity applications. Applications that use high interactivity commit implicitly as you navigate between rows of a list applet. You can edit any row of a list applet, and as you proceed these changes will be committed to the database.

Multi-Record Select List Applets

Multi-select list applets provide a way to select multiple items for a transaction. The check boxes in the left column are used to select the items. The Select All button allows the user to select all available records in the list. The Select action button selects all of the records that have been chosen for inclusion in the selection.

This feature is specific to standard interactivity applications. In applications that use high interactivity, multi-row selection is available in all list applets rendered using the `<swelist>` tag, except for pick applets. In these applets, multiple rows can be selected using the Control/Shift keys, as in any standard Windows application

The Siebel Web Engine supports the selection of multiple records in list applets for invoking methods that act on these selected records. With the HTML Client the selection of rows is done using check boxes that are placed on each row.

This is different from positioning the current record using the PositionOnRow control. You can have both the PositionOnRow control and Multiple Row Selection on the same list applet.

When you initially navigate to a list applet, the record on which the bus comp is positioned is automatically selected. Users can unselect this using the check box if desired. Unlike PositionOnRow, when you select rows using the check box there is no server round trip. The selected records are marked as selected on the bus comp only when a method is invoked on the applet. You can select records across multiple pages (that is, you can navigate using the Next and Previous controls and select records from different working sets).

By default, multirecord selection is not enabled for list applets. To enable this feature on list applets where this needs to be supported, set the new attribute of the List object in Tools called HTML Multi Row Select to TRUE.

To render the check boxes to select multiple rows in list applet templates, the tag `<swe:select-row>` is used. The syntax of this tag is:

```
<swe:select-row property="FormattedHtml" />
```

OR

```
<swe:select-row>
```

```
    <swe:this property="FormattedHtml" />
```

```
</swe:select-row>
```

When the property attribute is set to `FormattedHtml` in either the `<swe:select-row>` or `<swe:this>` tag, the check box will be rendered if the applet is enabled for multirecord selection in Tools. When `<swe:select-row>` tag is used without the property attribute, it acts as a conditional tag to show its body if the applet is enabled for multirecord selection.

By using this tag, you can create a generic list applet template that can be used with list applets that support multi-record selection and those that do not. In the list header, use the `<swe:select-row>` tag conditionally to put in a `<td>` for the header for the row selection check box column, and in the list body use the `<swe:select-row>` tag along with the `<swe:this>` tag conditionally to put in a `<td>` that contains the check box.

NOTE: You must place your list applet controls/list columns within a `<swe:form>` tag when you enable the multi-select feature, as any invoke method on the applet requires the form which contains the row selection check boxes to be submitted.

Controls that do not support invoking methods when multiple records are selected are not disabled when the user selects multiple records since there is no server call when selecting multiple records. Instead, when the control is activated a message will be shown to the user that the action cannot be performed when multiple records are selected

Displaying Totals of List Column Values

This feature supports the following:

- Simple summation of values in a list column
- Totals based on expressions defined at the business component field level

For example, the Revenue business component has the fields Quantity, Price, and Calculated Revenue. The field Calculated Revenue has an expression defined in its Calculated Value attribute as [Quantity]*[Price]. In a list applet based on this business component, you can show the total quantity and the total revenue. The total value for quantity is the sum of all quantity field values. The total value for revenue is the product of the totals of the quantity and price columns.

To configure a list applet to display totals

- 1** Set the Totals Displayed and Totals Required properties of the List object to TRUE.
- 2** Set the Total Required property of the specific list columns that need to be totaled to TRUE.
- 3** Set the Web Template used by the Base or Edit List applet Web template to Applet List Totals (Base/EditList).
- 4** Use the value Total for the Property attribute of the `<swe:control>` tag in the template file:

```
<swe:control id="XXX" property="Total"/>
```

or

```
<swe:control id="XXX">  
  <swe:this property="Total"/>
```

```
</swe:control>
```

When the Property attribute is set to Total, either in the `<swe:control>` tag or the `<swe:this>` tag, the total for the list column values is rendered if the list column is enabled for totals. If the list column is not enabled for totals, no output is generated. This property is valid only when the `<swe:control>` tag is mapped to a list column.

To enable expression-based totals

- 1 Check that the business component field to which the list column is mapped has an expression defined.
- 2 Set the Total Required attribute.
- 3 Add a user property named TotalAsExpr for the list column.

NOTE: Adding the user property is enough to evaluate the totals as an expression; the fields properties are ignored.

- 4 Use the value Total for the Property attribute of the `<swe:control>` tag in the template file as described in [Step 4](#) under “[To configure a list applet to display totals](#)” above.

You can also show totals in a separate applet. An example of showing the totals in a separate applet can be seen in the Quote Details View. A form applet appears below a list, which contains summations of columns within the list.

To show totals in a separate applet

- 1 Create a form applet and place it below the list applet in the view.
- 2 Create a field in the business component that sums a multivalued field using the calculated expression syntax `Sum([Multi Value Field])`.
- 3 Create in the business component a Multi Value Link object and a Multi Value Field object based on the link.

The Multi Value Link object references the business component that supports the list of values to be summed.

CAUTION: Never put a `Sum([Multi Value Field])` expression in a list column. This requires that a separate query be executed for each record in the list, which is a significant performance issue.

Multi-Value Group and Pick Applets

If a control or list column has an MVG applet configured in tools, SWE will have the following behavior:

- In Base mode, the field shows the primary value in the MVG. There is no link to pop up the MVG applet in this mode.
- In Edit, New or Edit List mode (provided the control or list column is editable), the field will show the primary value of the MVG as read-only text followed by a link to pop up the MVG applet. The style of the link to the MVG is configured using the `EditFieldCaption` and `EditFieldType` properties in the `cfg` file.
- When the link is activated, the MVG applet is rendered on a separate pop-up window.
- If the MVG applet has an Edit List type template defined in the repository, that template is used to render the applet. If not, the Base template is used. An error is generated if both Base and Edit List templates are not defined.
- The MVG applet behaves like any other list applet in the pop-up window. You can invoke methods like `EditRecord`, `AddRecord`, and `CreateRecord`. When these methods are invoked, the appropriate template is displayed in the current pop-up window. After the record is saved or selected, the MVG applet is again rendered in this window in the Base/Edit List mode.

Toggle Applets

Links to navigate between the toggle applets can be rendered as drop-down select controls or links, or tabs ([Figure 226](#)).



Figure 226. Toggle Picklist Applet

The toggle selection control can be rendered in any applet template using the new tag `<swe:togglebar>`. This tag works similarly to the `<swe:viewbar>` and `<swe:screenbar>` tags.

■ `<swe:togglebar>`

Usage:

```
<swe:togglebar type="xxx" property="zzz">
```

Attributes:

- **Type.** This can have one value, which is Select. If the type is set to Select, the togglebar will be rendered as a HTML Select control showing the set of applets that are available for the toggle. The applet titles are used as values in the select control.
- **Property.** This attribute is to be used only when the type is set to Select (will not have any effect in other cases). This attribute can have a value of FormattedHtml, in which case the HTML Select control is rendered. If this attribute is not specified, this tag acts as a conditional tag to show its contents if there are toggle applets defined. The `<swe:this>` tag will be used within the body of this tag in this case to render the select control.

If the applet does not have toggle applets defined, this tag and its contents are skipped.

When the type attribute is not set to Select, `<swe:togglelink>` and `<swe:togglename>` tags can be used within the body of the `<swe:togglebar>` tag to create the toggle links or tabs, similar to use of `<swe:viewlink>` and `<swe:viewname>` tags.

■ `<swe:togglelink>`

Usage:

```
<swe:togglelink state="xxx" property="yyy">
```

Attributes:

- **State.** This is optional. Can have values Active or Inactive. If state is Active, this tag will be used only if the current applet title being rendered is the currently active applet. If state is Inactive, this tag will be used only if the current applet title being rendered is not the currently active applet. If not specified, the tag is shown for all applets.
 - **Property.** This is optional. Can have only one value, FormattedHtml, which will output the HTML for creating a link to toggle to the applet. If this attribute is not specified, then no output is generated.
- `<swe:togglename>`

Usage:

```
<swe:togglename/>
```

Outputs the title of the applet.

Examples

To show the toggle applets as a select control:

```
<swe:togglebar type="Select" >
<table><tr>
  <td>
    <swe:control id="1" property="DisplayName">
  </td>
  <td>
    <swe:this property="FormattedHtml"/>
  </td>
</tr></table>
</swe:togglebar>
```

Here the control is used to create a label like Show: before the select control. To show the toggle applets as tabs or links:

```
<swe:togglebar>
```

```
<table><tr>
  <td>
    <swe:togglelink property="FormattedHtml">
      <swe:togglename>
    </swe:togglelink>
  </td>
</tr></table>
</swe:togglebar>
```


Tree Applets

The explorer-style (or tree) applet presents hierarchically structured information in an expandable tree control. The tree control is displayed in a frame on the left side of the applet content area. Detailed information for a selected tree node is displayed in the details applet in a frame to the right. The separate vertical frames allow the contents of the tree applet to be scrolled independently from the details applet. This is important because trees' structures can typically grow very large in length and width.

A tree applet in an explorer view is similar in operation to the Object Explorer and Object List Editor in Siebel Tools. The user may expand and collapse folders in the tree applet, and view the records in that folder in the list applet. The hierarchy displayed in the tree applet represents master-detail relationships between records of different business components. A tree applet in an explorer view uses the set of master-detail relationships implemented in the business object assigned to the view. The Opportunities Explorer View is illustrated in [Figure 227](#).

New	Name	Account	Primary	Revenue	Sales Stage	Close Date	Prior
*	30 Seats, Siebel Mid Revere National Cor	TSMYTHE		\$20,000.00	03 - Qualification	5/31/2002	
*	Call Center - 1000, Telstra Corp	TSMYTHE		\$3,000,000.00	01 - Prospecting	8/31/2001	
*	Call Center 700 Seats, Digithink	SPHILLIPS		\$1,125,000.00	04 - Opportunity	7/31/2001	
	Call Center 999	J.F. Davison Fund	SADMIN	\$0.00		7/21/2001	
	Marketing, DeVlyder Educational Solution	TSMYTHE		\$1,000,000.00	07 - Selected	4/30/2001	
	Siebel EAI, Unisys	InSuyan Inc	MSTERN	\$500,000.00	01 - Prospecting	8/31/2001	

Figure 227. Opportunities Explorer View

For example, when the user expands an opportunity by double-clicking, a set of folders appears hierarchically beneath it including Opportunities, Contacts, Partners, Quotes, Activities, Notes, and so on. When the user expands one of these child folders, a list of records appears of the corresponding business component. If the user expands the opportunity and then expands the Activities folder beneath it, the list of records displayed is the set of activity records for that opportunity. In the master-detail relationship between opportunities and activities, these activity records are detail records of the master opportunity record that was expanded. The user can also add or associate detail records of various kinds to particular master records.

NOTE: The architectural aspects of tree applets are described in the Tree Applets section of the [Chapter 13, “Special-Purpose Applets and Controls.”](#)

This section describes the configuration of the templates for the explorer applet.

Here is a sample view template for a view containing an explorer applet:

```
<!--View with tree applet on the left and list applet on the right-->
<table border="0" cellspacing="0" cellpadding="1" width="100%">
  <tr>
    <!-- Begin Tree Applet -->
    <td>
      <swe:applet id="1" hintText="Tree Applet"/>
    </td>
    <!-- Begin List Applet -->
    <td>
      <swe:applet-tree-list/>
    </td>
  </tr>
</table>
```

The `<swe:applet-tree-list>` tag that appears in this template provides a placeholder for a list applet that is displayed as a result of selecting or expanding a tree item (node). The applet that is rendered depends on the node that is currently selected.

Here is a sample applet template for an explorer applet. It displays the tree in a single-column table:

```
<TABLE BORDER=0 CELLPADDING=0 CELLSPACING=0>
  <TBODY>
    <swe:for-each-node>
      <TR VALIGN=top>
        <TD NOWRAP>
          <swe:for-each-indent>
            <swe:indent-img/>
          </swe:for-each-indent>
          <swe:node type="DisplayName">
            <swe:this property="FormattedHtml"/>
          </swe:node>
          <swe:node type="FieldValue">
            <swe:this property="FormattedHtml"/>
          </swe:node>
        </TD>
      </TR>
    </swe:for-each-node>
  </TBODY>
</TABLE>
```

A tree control can have repository tree nodes and field values as elements in the tree. The term *tree item* is used for a tree element regardless of whether it is a “root,” “branch,” or a “leaf” in the tree. A repository tree node is called a *tree node*. The `<swe:node>` tag specifies the placeholder for a tree item. For a tree node, the display name is shown, whereas for tree items, the field values are shown. In the example above, the `<swe:node>` tag with type `DisplayName` is ignored for tree items, and the `<swe:node>` with type `FieldValue` is ignored for tree nodes.

In order to display a tree, the logic iterates over each item of the tree in a top-down, depth-first fashion, and displays one item at a time. This is specified in the template using the `<swe:for-each-node>` tag.

Each tree item is indented to place the text in the correct indent level relative to the root using the `<swe:for-each-indent>` tag, and to display the expand/collapse mark, the text of the item, and the hyperlinks. The indentation is accomplished using a series of GIF images, or just white spaces (when in text-only mode). The expand/collapse mark and the item are displayed using images (or just text, in text-only mode), specified in the template using the `<swe:indent-img>` tag. The list applet associated with the currently selected tree node is displayed as part of the view.

Details about the various tags used in Tree Applet Templates are described below:

■ `<swe:for-each-node>`

Purpose:

Iterates over each visible item in the tree control in a top-down, “depth-first” fashion. This tag is used to display tree nodes and field values. The attributes are optional. If `Count` is not specified, the tag iterates over all nodes in the tree.

Attributes:

- **Count.** Specifies the number of times the tag should iterate its contents. This attribute is provided for situations where specific tree formatting is required.
- **StartValue.** The value at which the iteration starts. The tag starts the iteration by assigning this value to an internal iterator, and increments it by one for each iteration.

- `<swe:for-each-indent>`

Purpose:

Iterates over each level of a tree item. Used for creating indentation when displaying tree items.

Attributes: None.

- `<swe:indent-img>`

Purpose:

Provides a placeholder for a GIF image corresponding to a tree item's current indentation level. At each level, SWE determines which GIF file to use in the `` tag to output. The GIF images can be either a blank space or a vertical bar.

Attributes: None.

- `<swe:node>`

Purpose:

Provides a placeholder for an item in the tree. A tree item can be a repository tree node or a field value. The display name is printed if the tree item is a tree node. Otherwise, the field value is generated. The expand/collapse mark, the item's icon, and the links are also parts of a tree item. Depending on the configuration file settings, the expand/collapse mark is shown as either a GIF image or text. The expand/collapse mark is only shown for tree items with child items. There are two links associated with each item. There is a link for the `+/-` mark to expand or collapse the item and a link for the item image for selecting the item (or for going to next or previous workset). The item selection allows the user to access the list applet associated with the tree node. This tag should use `<swe:this>` as a child tag.

Attributes:

- **type.** Set to "DisplayName" or "FieldValue". Outputs the repository tree node's Display Name if "DisplayName." Otherwise, outputs field values.

Configuration File Parameters

A tree control consists of reusable graphic elements and text obtained from business component records, as shown in [Figure 228](#).

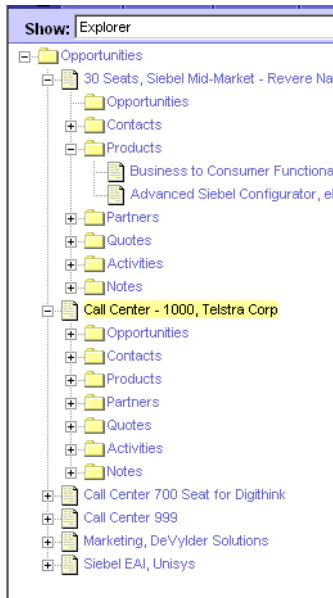


Figure 228. Tree Control

The text is obtained from business components, as defined in the Tree and Tree Node object types in the repository. The graphic elements (expansion and contraction boxes, elbows, folder symbols and so on) are defined in the configuration file in the [SWE] section. Configuration file parameters are specified to customize the appearance of the folder and document symbols, expand and collapse marks, elbows, spacers, and so on. The syntax of a configuration file parameter line for defining a graphic is as follows:

```
parameter_name = <img param1 param2 etc.>
```

For example:

```
TreeNodeCollapseCaption = "<img src='images/tree_collapse.gif'  
alt='- ' border=0 align=left vspace=0 hspace=0>"
```

A text replacement for an image, for use by text-only browsers, is specified using the `alt=` parameter in the `` tag.

Four parameters are also available for configuring the display of text obtained from field values. These are the `TreeNodeFontStyle`, `TreeNodeFontSize`, `TreeNodeSelectBgColor`, and `TreeNodeSelectFgColor` parameters. The syntax for these is:

```
parameter_name = value
```

The term *caption* as used in the parameter names is a bit misleading. By caption, what is actually meant is icon or graphic. The so-called caption precedes the text that is generated from field values, or precedes another so-called caption. The supported caption graphic and text style parameters are listed below by category.

Elbows and Trees

- `TreeNodeCollapseElbowCaption`
- `TreeNodeCollapseTeeCaption`
- `TreeNodeElbowCaption`
- `TreeNodeExpandElbowCaption`
- `TreeNodeExpandTeeCaption`
- `TreeNodeTeeCaption`

Root, Leaf, and Open/Closed Folder Icons

- `TreeNodeCloseFolderCaption`.
- `TreeNodeLeafCaption`.
- `TreeNodeOpenFolderCaption`—Open folder with the dangling line.
- `TreeNodeOpenFolder2Caption`—Open folder without the dangling line.
- `TreeNodeRootCaption`.
- `TreeNodeArrowDownCaption`—This icon indicates that there are additional records not shown below, and when clicked, displays the next group.
- `TreeNodeArrowUpCaption`—This indicates that there are additional records not shown above.

Indentation Graphics

- `TreeNodeBarCaption`.
- `TreeNodeSpaceCaption`.

Text Style Parameters

- `TreeNodeFontStyle`. Defaults to MS Sans Serif,Arial,Helvetica.
- `TreeNodeFontSize`. Defaults to 1.
- `TreeNodeSelectBgColor`. Defaults to #000080.
- `TreeNodeSelectFgColor`. Defaults to #ffffff.m

Chart Applets

Chart Applets display business component data as different types of charts and graphs. Templates for charts contain a handful of `swe:control` tags to map the Chart Control (id = 599) in the standard configuration) and the various controls for switching between different chart types, and so on. A typical chart template is shown below:

```
<table width="98%" cellspacing="0" cellpadding="0" border="0"
align="center">

<swe:form>

    ...

<table width="100%" valign="top" align="center">

    <swe:togglebar type="Select">

<tr>

    <td>

        <swe:control id="2" property="DisplayName" />

    </td>

    <td>

        <swe:this property="FormattedHtml" />

    </td>

</tr>

</table>

</swe:form>

</table>
```



```
        </td>
    </tr>
</swe:togglebar>
</table>
...
<table class="AppletBack" width="100%" border="0">
    <tr>
        <td align="center">
            <swe:control id="599" property="FormattedHtml"
                hintText="Chart" />
        </td>
    </tr>
</table>
...
</swe:form>
</table>
```

Catalog-Style List Applets and Rich Lists

This feature supports a catalog-like layout for views with master-detail applets. Records from the master applet and the detail applet can be shown interwoven with each other. This allows the creation of the layout like the one shown below in [Figure 229](#):



Figure 229. Master-Detail Applet

In this case the bullet items under Portable Music are from records in the master applet. The values below it are records from the detail applet for that record in the master applet.

To create this layout, the master and detail applets are configured to be list applets. The master applet will be called a root level applet. It is possible to show more than one set of master-detail relationships within a view (that is, there could be more than one root level applet). To define the relationship between the applets, the new Position attribute of the View Web Template Item object type is used. The position attribute works similarly to the Position attribute of the Tree Node object type. The root level applets will have position values like 1, 2, and so on. For the applet with position 1, its immediate child applets will be assigned position values 1.1, 1.2, and so on. It is possible to define a third level applets with position 1.1.1, 1.1.2, and so on (that is, these are the child applets of the applet with position 1.1).

In the View Web Template Item object definition, only the root level applets are mapped to `<swc:applet>` tags in the view template. The other applets in the view defined in the View Web Template Item object are not assigned an Id value. The layout of these non-root applets are not specified in the view template, but are specified in the applet template of the root level applets. The following new tags are used to specify this layout. Only applets in the base mode in this layout are supported.

- `swe:for-each-child`

Usage:

```
<swe:for-each-child> ... </swe:for-each-child>
```

Purpose: This tag iterates over each of the child applets defined for this applet (based on the Item Identified in the View Web Template Item object of the view to which the applet belongs). This tag can be used only in the base template of an applet. If the applet does not have any child applets, this tag is skipped.

- `swe:child-applet`

Usage:

```
<swe:child-applet/>
```

Purpose:

This tag is used to place the child applet within the parent applet. The base template of the child applet is used to render the child applet at the point where this tag is placed.

Example

This section presents a master-detail applet relationship with the master applet being Category Items List Applet and the detail being Sub Category Items List Applet. The View Web Template Item of the view that contains this applet has the following values (Table 57):

Table 57. View Web Template Item Properties

Item Identifier	Applet	Applet Mode	Position
101	Category Items List Applet	Base	1
	Sub Category Items List Applet	Base	1.1

The base template for the Category Items List Applet will have the following table definition:

```
<table>
<swe:for-each-row>
```

```
<tr>
  <td>
    <swe:control id = "5001" /> <!-- field value like "Small
    Business" -->
  </td>
  <td>
    <swe:for-each-child>
      <swe:child-applet> <!-- Show the child applet -->
    </swe:for-each-child>
  </td>
</tr>
</swe:for-each-row>
</table>
```

The base template for the Sub Category Items List Applet will have the following:

```
<table><tr>
<swe:for-each-row>
  <td>
    <swe:control id="5001"/> <!-- field value like "Desktop" -->
  </td>
</swe:for-each-row>
</tr></table>
```

NOTE: Set the HTML Number of Rows property of the Sub Category Items List Applet to the number of values you want to show under each category value. To allow drilldown from the category and sub-category values, configure the appropriate drilldown objects.

Toolbars and Menus

Toolbars and menus allows the user to initiate various actions. Toolbars appear in their own frame near the top of the application in the browser window, and the application menus (File, View and Help) appear just beneath the primary tab (first navigational level) bar, as shown in [Figure 230](#).

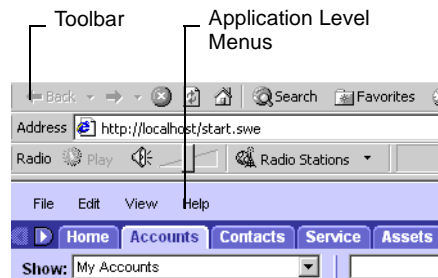


Figure 230. Primary Tab Bar

The applet-level menus are invoked from the applet menu button, in the control banner at the top of an applet. This is illustrated in [Figure 231](#) where the menu button is in the left corner of the control banner.



Figure 231. Applet Menu Button

Clicking on a toolbar icon or menu item is normally translated into a call to an invoke method, which may reside in a service on the browser or server, or in classes in the browser application or server infrastructure (applet or business component classes). The toolbar icon or menu item is configured to target a method name, a method handler, and optionally a service.

Application-level items (which include both toolbar icons and application-level menus) are implemented through the use of Command object definitions in Tools, which are then mapped to Toolbar Item or Menu Item object definitions. Applet-level menus do not use Command object definitions, but the set of properties used for targeting the method are essentially the same as those found in the Command object type.

In SWE templates, the `<swe:toolbar>` tag specifies a named toolbar (where the name corresponds to the Name property in the Toolbar object definition in the repository), and the `<swe:toolbaritem>` tag between the toolbar start and end tags recursively retrieves all of the toolbar items for that toolbar from the repository.

Toolbar Template Configuration

Three types of toolbars are supported: regular and Java applet. Regular toolbars are displayed as HTML toolbars in standard interactivity applications and as extensible JavaScript toolbars in high interactivity applications.

HTML toolbars reside in the topmost frame in the application template, which is set aside for this purpose. JavaScript toolbar objects reside in the JSSApplication hidden frame, which usually does not reload during the application life cycle. Therefore, they are not redrawn when there is a page refresh. The UI part of the JavaScript toolbar resides in a visible HTML frame (recommended to be a persistent frame that reloads infrequently) and redraws when the HTML frame reloads.

An additional frame beneath these is specified for Java applet toolbars in Siebel Call Center and similar applications using CTI. If no Java applet toolbar is used, this frame is omitted.

HTML and JavaScript Toolbars

For an HTML or JavaScript toolbar, add the following to the SWT file:

```
<swe:toolbar name=xxx> // where xxx is the name of toolbar in the
repository.

// any HTML stuff here...
```

```
<swe:toolbaritem>
    // any HTML stuff here...
</swe:toolbar>
```

NOTE: For combobox items, the command has to be targeted to a service.

Java Toolbars

For a Java toolbar, add the following to the SWT file:

```
<swe:toolbar name="xxx" javaapplet="true" />
```

The Java applet invokes the ShellUIInit method on the command target service when it tries to initialize. It invokes ShellUIExit when it exits. There is a set of communication protocols defined for the communication between the Java Applet and the service.

The toolbar is implemented as a Java Applet (including all the toolbar controls and the threads interacting with the server).

The full syntax specifications for the `<swe:toolbar>` and `<swe:toolbaritem>` tags are below:

■ `<swe:toolbar>`

Purpose:

In SWE templates, the `<swe:toolbar>` tag specifies a named toolbar (where the name corresponds to the Name property in the Toolbar object definition in the repository), and the `<swe:toolbaritem>` tag between the toolbar start and end tags recursively retrieves all of the toolbar items for that toolbar from the repository. Siebel eBusiness Applications currently support two types of toolbars: HTML toolbars and Java applet toolbars, as specified in the `javaapplet` attribute.

Usage:

```
<swe:toolbar name="XXX" javaapplet="true/false" width="XXX"
height="XXX" />
```

Attributes:

- **name.** The name of the toolbar as defined in Tools.
- **javaapplet.** This should be set to true for java toolbar, and false for HTML toolbar.
- **width.** Width of the toolbar in pixels.
- **height.** Height of the toolbar in pixels.
- `<swe:toolbaritem>`

Usage: `<swe:toolbaritem>`

Attributes: None.

Menu Template Configuration

A menu, in SWE templates, is a button or link that activates a drop-down list of menu selection items when clicked by the user. There are application-level and applet-level menu buttons/links, as described previously. The `<swe:menu>` tag will render all menus that are appropriate at the template level where it occurs—application (container page) or applet. Menus are shown only when running in the High Interactive mode. An applet menu is rendered as an icon button, generally placed to the left of the other buttons such as Edit and Delete. For a description of the configuration of the set of menu options for an applet menu, see [Chapter 9, “Logical User Interface Objects Layer.”](#) At runtime, the set of menu options for a given applet are generated from the .srf file. Note that for applet level menus, the tag must be specified in an applet template.

A sample of the template for rendering an applet's buttons, including the menu button, appears below (from CCFormButtons.swt):

```
<!-- Buttons (Edit, Delete, Optional, Optional, Optional) --->
<!-- Menu,179 -->
    <td valign="middle" nowrap>
        <swe:menu/>
```



```

    </td>

    <td valign="middle"></td>

<!-- EditRecord -->
<swe:control id="132">

    <td valign="middle" nowrap>

        <swe:this property="FormattedHtml" hintText="Edit"
        hintMapType="Control"/>

    </td>

    <td>&nbsp;</td>

</swe:control>

(and so on...)
```

For the application level menu, the `<swe:menu>` tag can be specified in any template other than an applet template. The set of menus is rendered from the one `<swe:menu>` tag, based on the Menu and Menu Item object definitions in the repository. The Menu object definition that is used is the one pointed to by the Menu property in the Application object definition. This Menu object definition specifies a set of top-level menu names and menu options within each top-level menu. The Profile and Logout menu options are automatically provided in all applications, and do not have to be defined within the application's Menu object definition and its children.

The following code sample (from `CCFrameBanner.swt`) illustrates the inclusion of the `<swe:menu>` tag at the start of the definition for a banner:

```

<!--Start Banner-->
<swe:menu/>

<table class="banner" cellpadding='0' cellspacing='0' border='0'>
<tr>
    <td width="50%">

        
```

```
</td>

<td width="50%">

</td>

</tr>
```

The syntax for the `<swe:menu>` tag appears as follows:

■ `<swe:menu>`

Purpose:

Renders menu buttons or links for all menus defined for the relevant entity, either an application or an applet. For an application, one button or link is rendered for each top-level menu defined for the application (in its associated Menu object definition and children). For an applet, one button is rendered, the applet menu button.

Usage:

```
<swe:menu type="XXX" bitmap="XXX" width="XXX" height="XXX"
bgcolor="XXX" fgcolor="XXX" />
```

Attributes:

- **type.** The type can be set to Default (which is the default value if this attribute is not specified) or Button. If set to Default the menu is rendered showing the top level menu entries (like File Edit Help). If set to Button, a button is created that, when activated, shows a drop-down menu with the top level menu entries.
- **bitmap.** This attribute is used only when the Type attribute is set to Button. This attribute is used to specify the name of a bitmap object to be used as the label for the button. This bitmap is defined in Tools under the bitmap category HTML Control Icons.
- **width.** This attribute is used to specify the width of the menu in pixels.
- **height.** This attribute is used to specify the height of the menu in pixels.

- **bgcolor.** This attribute is used to specify the background color of the menu. The color should be specified using the hexadecimal triplet format used in HTML like #FFFFFF.
- **fgcolor.** This attribute is used to specify the foreground color of the menu. The color should be specified using the hexadecimal triplet format.

Thread Bar

The thread bar is used to track user navigation among the views. A thread bar in HTML text format has been implemented. An example of the thread bar is as follows:

```
Home > Consumer:PCs > PCs:Laptops > Laptops:Pentium III
```

Home, Consumer:PCs, and so on are the thread buttons. The thread buttons are displayed in title: value format, and either title or value can be omitted when appropriate. The thread button may contain a hyperlink, which leads the user to a previous page. The thread buttons are separated by separators— “>” in the above example.

A thread button may have a hyperlink that leads the user to a previous page. The hyperlink requires a new SWE Command: GotoBookmarkView. The hyperlink for each thread button should contain at least the following parameters:

```
SWECmd=GotoBookmarkView&SWEBMCount=2SWECCount =3
```

The SWEBMCount = 2 indicates that bookmark #2 will be used to create the view. SWECCount = 3 is the bookmark ID for the current view. With the definition of the swe tags and thread link format, a thread button for account A.K. Parker will be translated into HTML format as:

```
<a href = " www.siebel.com/  
start.swe?SWECmd=GotoBookmarkView&SWEBMCount=2& SWECCount=3>  
Account: AK Parker </a>
```

A new bookmark is created when the user clicks the thread button and brings back a bookmarked view. The bookmark ID for the new view is the current swe count (the count passed to the server in the request) increased by 1.

Bookmark deletion policy is not modified with the above bookmark ID assignment policy. By default, the system keeps the most recently created 20 bookmarks and deletes previous ones. If the swe count in the user request is less than the swe count on the server side, all the bookmarks with a swe count larger than what is in the user request is deleted.

The behavior of the HTML threadbar is summarized below:

- When a new screen is requested, a new thread is created to replace the current thread.
- When a view button is clicked, the last thread step is replaced by that of the new view requested.
- When the user follows a drilldown link, a new step is appended on the thread bar for the view requested.
- When a thread button is clicked, all the thread buttons to the right of it are all deleted.
- Some views may not have a thread applet or thread field defined. Showing these views do not cause the thread button to be updated.

When a thread button is clicked, the thread proceeds to the step view indicated by `SWEBMCount`.

The following three swe tags are defined to create an HTML thread bar. The usage of these swe tags is very similar to that of the screen bar and view bar tags.

- `<swe:threadbar>` Indicates the start and finish of the thread bar section.
- `<swe:threadlink>` Indicates the definition of a thread button on the thread bar. This tag has two properties defined:
 - `FormattedHtml` property. Indicates that HTML hyperlink should be included.
 - `Title` property. Indicates that the title / value pair of the thread button should be displayed.
- `<swe:stepseparator>` Specifies the symbol used to separate thread buttons.

The `<swe:threadlink>` and `<swe:stepseparator>` tags should only be used within the `<swe:threadbar>` tag.

To use a thread bar, insert thread bar definitions into an appropriate SWT file by using the tags defined above. An example is given below:

```

<!-- Begin Threadbar section -->
<table class="Theadbar" width=100% border="0" cellspacing="0"
cellpadding="0">
<tr valign="left">
  <td nowrap bgcolor="#6666CC" width=110>
    
  </td>
  <td width=99%>
    <swe:threadbar>
      
      <swe:threadlink property="FormattedHtml">
        <font color="#000000"> <span >&nbsp;<no><swe:this
property="Title"/></no>&nbsp;</span> </font>
      </swe:threadlink>
      <swe:stepseparator>&gt;</swe:stepseparator>
    </swe:threadbar>
    
  </td>
</tr>
</table>
<!-- End Threadbar section -->

```

This creates a thread bar as shown below:

Home > Consumer:PCs > PCs:Laptops

For applications without frames, put the definition in a container page such as CCPageContainer.swt; for applications with frames, insert it in the “Viewbar” frame swt file or View frame swt file.

Special Behavior Supported by Templates **16**

This chapter describes some of the special behavior supported by templates, such as Search and Find, favorites (predefined queries), and conditional mappings for applets.

Search and Find Configuration in SWE Templates

The Siebel Web client supports a unified search model that merges the functions of search, find, and query in order to provide users with multiple methods of locating records. Depending on business requirements, various application-level searching/querying and applet-level querying are available to the customer to configure and add. See [“Screens” on page 502](#) for further information about this feature.

Various specialized SWE tags are used in the Search and Find applet, and the Results applet. They fall into two groups.

Search and Find Applet Tags

These tags are used to display the Basic and Advanced Search or Find applets. The tags are `<swe:srchCategoryList>`, `<swe:srchCategory>`, `<swe:srchCategoryText>`, and `<swe:srchCategoryControl>`.

Example:

```
<swe:srchCategoryList>
  <swe:srchCategory>
    <td><swe:srchCategoryText/></td>
    <td><swe:srchCategoryControl/></td>
  </swe:srchCategory>
```

```
</swe:srchCategoryList>
```

The syntax for each tag is described as follows:

■ `<swe:srchCategoryList>`

Purpose:

An iterator tag that encloses all the search categories that need to be displayed.

Usage:

```
<swe:srchCategoryList> ... </swe:srchCategoryList>
```

Encloses all three tags described below. This tag also establishes a context.

■ `<swe:srchCategory>`

Purpose:

Represents a search category object.

Usage:

```
<swe:srchCategory> ... </swe:srchCategory>
```

Encloses the two tags described below.

■ `<swe:srchCategoryText>`

Purpose:

Displays the “display name” of the search category.

Usage:

```
<swe:srchCategoryText/>
```

Can be called only within the context of a “srchCategory.”

■ `<swe:srchCategoryControl>`

Purpose:

Displays the control of the search category. In the case of the Advanced Search, it is a check box.

Usage:

```
<swe:srchCategoryControl/>
```

Can be called only within the context of a “srchCategory.”

Results Applet Tags

These tags are used to display the Search/Find results list applet. They appear in the CCListBodySearchResults.swt and dCCListBodySearchResults.swt templates, and include the following tags: `<swe:srchResultFieldList>`, `<swe:srchResultField>`, and `<swe:this>`.

Example:

```
<swe:srchResultFieldList>
  <swe:srchResultField><td align="swe:this.TextAlignment"
    class="Row"><swe:this property="FormattedHtml" />&nbsp;</td>
  </swe:srchResultField>
</swe:srchResultFieldList>
```

The syntax for these tags is described as follows:

■ `<swe:srchResultFieldList>`

Purpose:

An iterator tag that encloses all the search result fields that are defined in Tools under a Search Engine Object. The result fields are created dynamically in the buscomp and then displayed on the applet.

Usage:

```
<swe:srchResultFieldList> ... </swe:srchResultFieldList>
```

Encloses both tags described below. This tag also establishes a context.

■ `<swe:srchResultField>`

Purpose:

Represents a result field object.

Usage:

```
<swe:srchResultField> ... </swe:srchResultField>
```

Encloses the tag described below and can be called only within the context of the `srchResultFieldList`.

■ `<swe:this>`

Purpose:

Depending on the `property = setting`, retrieves either the text alignment setting or the value for the current result field.

Usage:

```
<swe:this/>
```

Attributes:

- **property="TextAlignment."** Retrieves the text alignment property for the result field from the Tools object "Search Definition - Custom result Field."
- **property="FormattedHtml."** Retrieves the value for the current result field from the results obtained by executing the search on the search adapter.

Behavior varies depending on the enclosing context.

Favorites (Predefined Queries)

Configuration of the predefined query feature has configuration file aspects, SWE tag aspects, and menu/toolbar command aspects.

The SWE tag `<swe:PDQbar>` is used to implement PDQ functionality. It has no parameters and can be located anywhere in the application. The user selects the query to be executed. The only thing the template developer needs to explicitly provide besides the `<swe:PDQbar>` tag is the “Favorites” label to the left of it. Ideally, the Favorites label should be implemented as a control rather than HTML text, so that it will be translated for localized or multilingual applications.

The `<swe:pdqbar>` tag is not required to be in the view frame. The `<swe:pdqbar>` tag should be placed either in the view frame or the view bar frame in applications that use HTML frames.

Query Management Commands

Users add their named queries into the combo box by means of the query management commands available as invoke method calls through the base applet classes. These are made available to the user as menu options or toolbar buttons. The following commands are supported:

- **New.** `SWEMthdNewQueryE`. This command places the applet in new query mode.
- **Refine.** `SWEMthdRefineQueryE`. This command places the applet in query-refinement mode.
- **Save.** `SWEMthdSaveQueryE`. This command saves the current query as a named query using its current name.
- **Save As.** `SWEMthdSaveQueryAsE`. This command opens up a dialog box to save the current query as a named query using a user-specified name.

- **Delete.** SWEMthdDeleteQueryE. This command opens up a dialog box to delete one of the named queries.

NOTE: Siebel Systems does not recommend implementing an Edit button for the predefined query (PDQ) feature. To implement the Edit button you would need to have it call the Refine invoke method. However, there can be problems associated with implementing an Edit button in a multiview environment, in which there is no way to determine the active view.

Conditional Tags

The following sections describe conditional tags in Siebel Web templates.

SWE Conditional Tags

The SWE framework supports the following conditional tags.

- `<swe:if>`

Provides a simple conditional branching capability.

Usage:

```
<swe:if condition="xxx"> ... </swe:if>
```

Attributes:

- **Condition.** The condition to check for. If the condition evaluates to TRUE, the body of the `<swe:if>` tag is processed. If the condition evaluates to FALSE, the body of the tag is skipped.

NOTE: This tag does not provide an “else” capability like the if tags in programming languages. To get that behavior use the tags `<swe:switch>`, `<swe:case>`, and `<swe:default>` described below.

- `<swe:switch>`, `<swe:case>`, and `<swe:default>`

These three tags are used together to provide a conditional branching capability similar to the `switch`, `case`, and `default` statements in JavaScript. The `<swe:switch>` is a container tag for the `<swe:case>` and `<swe:default>` tags. Anything other than `<swe:case>` and `<swe:default>` within the body of the `<swe:switch>` tag is ignored. The condition to check is specified as an attribute of the `<swe:case>` tag. The `<swe:case>` tags are checked starting from the first `<swe:case>` tag. If any of the `<swe:case>` tags satisfies the condition, the other `<swe:case>` tags and the `<swe:default>` tags are skipped. If none of the `<swe:case>` tags satisfy their condition, the body of the `<swe:default>` tag is processed. There should only be one `<swe:default>` tag within the body of a `<swe:switch>` tag.

Usage:

```
<swe:switch>
  <swe:case condition="xxx">
    ...
  </swe:case>
  <swe:case condition="yyy">
    ...
  </swe:case>
  <swe:default>
    ...
  </swe:default>
</swe:switch>
```

Attributes:

- **Condition.** Supported only in the `<swe:case>` tag. If the condition evaluates to TRUE, the body of the `<swe:case>` tag is processed. Any subsequent `<swe:case>` tags within the `<swe:switch>` tag is skipped without checking their associated conditions. If the condition evaluates to FALSE, the body of the tag is skipped.

The SWE framework supports a limited set of conditions that can be checked using the conditional tags. These are described in the following sections.

- `<swe:if-var>`

The `<swe:if-var name="[value]">` tag is used within applet templates to conditionally express its body based on a variable set in a parent view template. When an applet is associated with a view, the applet's template(s) acts as a child of the view's template for the purposes of the `swe:if-var` tag. The applet placeholder in the view template must specify a variable for the `swe:if-var` tag in the child applet template to evaluate. The `swe:if-var` expression returns true or false depending on whether the variable it is evaluating is a property of the `<swe:applet>` tag in the corresponding view template. This construct is useful for conditionally displaying parts of an applet depending on its position within a view.

Figure 232 shows a diagram of the relationships.

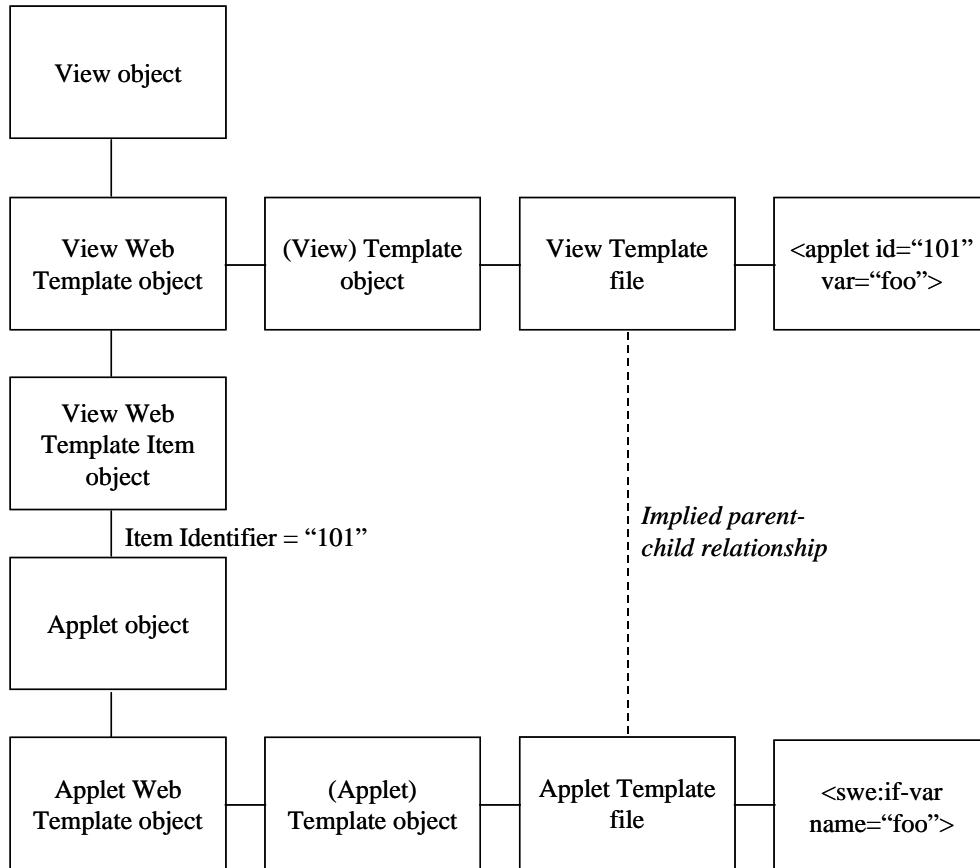


Figure 232. Object Relationships

Consider an example where a view uses a template that contains the following tags:

```
<swe:applet hintMapType="Applet" id="1" property="FormattedHtml"
hintText="Applet" var="Parent"/>
```

```
<swe:applet hintMapType="Applet" id="2" property="FormattedHtml"
hintText="Applet" var="Child"/>
```

The view object also references an applet (through a view Web template item) whose template includes the following tags:

```
<swe:if-var name="Parent">
    <td valign="middle" nowrap>
        <swe:menu type="Button" bitmap="MenuBttn" width="38"
            height="15" bgcolor="gray" fgcolor="blue"/>
    </td>
</swe:if-var>
<swe:if-var name="Child">
    <td valign="middle" nowrap>
        <swe:menu type="Button" bitmap="MenuBttn" width="38"
            height="15" bgcolor="gray" fgcolor="red"/>
    </td>
</swe:if-var>
```

If the user drags and drops the applet into the placeholder in the view template with an `id = 1`, the first `swe:if-var` condition will return `TRUE` and the second will return `FALSE`. This is because the `<swe:applet>` placeholder with an `id = 1` has its `var` property set to "Parent." As a result, the button menu will be displayed with a foreground color of blue. By contrast, if the user had mapped the applet to the placeholder represented by `<swe:applet id="2">`, the reverse would be true, and the button menu will be displayed with a foreground color of red.

Designing Browser Group-Specific Templates

The SWE framework supports a set of browser group-related conditions that can be checked in the Web templates using SWE conditional tags. This allows showing different sections of the template based on which browser is used to access the application.

The information about the supported user agents is defined in the Web Browser Administration views accessible through the Web Client out-of-the-box. Siebel applications have a series of predefined browsers and their associated capabilities. Examples of capabilities include items such as “FrameSupport,” which indicates that a browser can support ActiveX controls. Customers can modify the records that define these browsers and their capabilities as new browsers or new versions of existing browsers are introduced. Details on how to do this are provided in the *Applications Administration Guide*.

Given below is an example of the capabilities associated with Microsoft Internet Explorer:

```
[ IE 5.0 ]

CookiesAllowed=TRUE

HighInteract=TRUE

ActiveX=TRUE

Browser=IE

Version=5

DefaultMarkup=HTML

VBScript=TRUE

JavaScript=TRUE

JavaApplets=TRUE

User-Agent=Mozilla/4.0 (compatible; MSIE 5.0

SynchExternalContent=TRUE

FramesSupport=TRUE

TablesSupport=TRUE
```

Below is an example of the extended sections for Microsoft Internet Explorer:

```
[MSIE 5.0]

User-Agent=Mozilla/4.0 (compatible; MSIE 5.0

Parent=IE 5.0
```

Special Behavior Supported by Templates

Conditional Tags

```
Accept=image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,  
application/vnd.ms-powerpoint, application/vnd.ms-excel,  
application/msword, */*
```

```
[MSIE 5.5]
```

```
User-Agent=Mozilla/4.0 (compatible; MSIE 5.5
```

```
Parent=IE 5.0
```

```
Version=5.5
```

```
Accept=image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,  
application/vnd.ms-powerpoint, application/vnd.ms-excel,  
application/msword, */*
```

```
XML=TRUE
```

```
WAP=FALSE
```

```
StyleSheets=TRUE
```

```
JavaScriptVer=1.3
```

```
DHTML=TRUE
```

The following conditions can be used to check browser related information.

NOTE: Practically speaking the term “User Agent” is a synonym for “Browser.” Its usage comes from the User Agent header property of an HTTP request which provides a unique identifier for the type of client that is making the request, such as “Mozilla/4.0 (compatible; MSIE 5.0; Windows NT 4.0)” for Microsoft Internet Explorer 5.0.

The conditions are specified in the format <service>, <method>, <args> ...

■ **Service:** Web Engine User Agent

Method: IsUserAgent

Args: UserAgent: < A User Agent name defined in the UA.INI file >

Purpose: Checks for a particular User Agent.

Example:

```
<swe:if condition="Web Engine User Agent, IsUserAgent,
'UserAgent:MSIE 5.5'">
    ...
</swe:if>
```

The condition evaluates to TRUE for the Microsoft Internet Explorer 5.5 browser and FALSE for all other browsers.

- **Service:** Web Engine User Agent

Method: TestCapability

Args: Capability Name:Capability Value

Purpose: Check for specific user agent capabilities. When more than one capability is provided as an argument, the condition evaluates to TRUE when the user agent has all these capabilities (AND operation).

Example:

```
<swe:if condition="Web Engine User Agent, TestCapability,
'JavaScript:TRUE', 'JavaApplets:TRUE'">
    ...
</swe:if>
```

The condition evaluates to TRUE for any user agent that supports JavaScript and Java Applets in the browser.

Conditional Mappings for Applets

Conditional mappings are settings in the repository that modify the appearance of an applet depending on the current target browser group, application, language, and more/less mode. These are implemented with properties in the Applet Web Template object type, and its children and grandchildren, and are set using certain UI features in the Tools environment.

The following features support conditional control mappings in applets:

- **Browser Specific.** Based on the target browser group or virtual browser group the user is using.
- **Application Specific.** Based on the application the user is running.
- **Language Specific.** Based on the language/locale in which the application is running.
- **More/Less Mode Specific.** Based on whether the applet is being displayed by the user in the More mode or the Less mode. The user can toggle between these modes to display more or fewer controls or list columns.

Conditional control mappings are added and removed in Tools when working in the Applet Web Editor by using picklist controls in the toolbar that specify the affected target browser group, application(s), language(s) and more/less mode. Conditional mappings are interpreted by SWE at runtime as it interprets and parses the template.

Each of the functional areas of conditional mapping is described in a subsection.

Browser-Specific Mappings

As described above, different browsers can have different associated capabilities as defined in the Web Browser Administration view. Differences in the abilities of particular browsers to use features such as frames and JavaScript may necessitate different applet layouts for different classes of browsers. The browser group-specific mappings feature allows this kind of browser-specific layout customization.

A Target Browser combo box appears in the toolbar when the View > Toolbars > Configuration Context option is activated. The first item in the combo box is Target Browser Config, which activates a dialog box for selecting browsers.

The Target Browser Configuration dialog box lists all of the browsers in the Available browsers selection box at the upper left. You select the ones you are interested in working with by moving the names of these browsers into the Selected browsers for layout editing section at right. You can also view various features of the selected browser in the Capability Name and Value box in the lower half of the dialog box. The boxes in the dialog box include the following:

- Available browsers: List of available browser groups.

- Selected browsers for layout editing: Specifies which browser groups are affected by subsequent layout editing in the Web Layout Editor.
- Capability Name and Value: Specifies what capabilities or properties the currently selected virtual browser group has.

The target browser group determines how conditional template tags are expressed in the layout editor.

Inside the template are SWE:IF tags that conditionally execute blocks of code. (Markup is included.) At edit time you see the applet the way it would show up for the particular browser you have chosen. At run time, the conditional sections in the template are executed appropriately for the current browser.

For example, a user may associate a template to a view that contains the following tags:

```
<swe:if condition="Web Engine User Agent, TestCapability,
'FrameSupport:TRUE' ">

  <swe:frameset htmlAttr="cols='33%,66%' border='1'
frameborder='Yes' ">

    <swe:frame type="Applet" htmlAttr="marginheight='0'
marginwidth='0' scrolling='Auto' ">

      <swe:applet id="101" hintText="Applet" var="Parent">

        <swe:this property="FormattedHtml"/>

      </swe:applet>

    </swe:frame>

  </swe:frameset>

</swe:if>
```

If the user opens a view in the Web Layout Editor when his or her Target Browser is set to IE 5.0, the user will see a placeholder for the applet in the frame with an underlying identifier of 101. The user can then drag and drop a particular applet to the placeholder. This is because IE 5.0 has a FrameSupport capability with a value of TRUE. However, if the user's Target Browser is set to IE 1.5, he or she will simply not see the placeholder in the layout editor since the FrameSupport capability for IE 1.5 is set to FALSE.

Application-Specific Mappings

Application-specific controls or list column mappings specify that the corresponding control is active or not when a particular application is active. This provides the means to display or remove controls or list columns at runtime based on the application that is running. They can be repositioned for individual applications.

The application setting of the Application combo box in the Configuration Context toolbar determines the setting applied to control mappings that are subsequently added or removed. By default, the layout editor is in All Applications mode, which leaves the controls that are added or deleted during the session unmodified. A specific application name can instead be chosen from the combo box, which places the layout editor in single-application mode, with the effect that controls that are added or deleted have that effect only for the selected application.

A conditional application-specific setting for a control is implemented with the Expression property in the Applet Web Template Item object definition for the control. The Expression property functions as a search specification or query condition, limiting the display of the control to those applications that match the expression condition. The Expression property is normally blank, which means “unrestricted,” that is, the control appears in all applications. A single application name in the property value, such as eSales, restricts the control to appearing only in the specified application. A negation expression, such as NOT eSales, specifies that the control appears in all applications except the negated one.

Normally, the Expression value is not entered directly. Instead, the developer sets the Application combo box in the toolbar to the application to work on, and adds and removes controls or list columns through drag-and-drop. When a user adds a control with an application selected specifies that control only appears for the selected application (the application name is automatically entered in the Expression property for the control).

NOTE: This feature could be used to add a new button that is only required in a particular application.

If you choose a different selection in the Application combo box during an editing session, the layout window changes to reflect the set of controls that is specified for that application. Namely, the controls appearing would be those specified for All Applications, plus those specified for the current application, less those negated for this application.

Unlike the target browser specific mappings, wizards do not affect application-specific mappings. If you use wizards to create something, it always gets created for all applications.

More/Less Mode-Specific Mappings

This feature provides the ability for an applet to display initially with a limited set of fields, and expand to display a larger number of fields when the user clicks a More button. The applet has two modes, a Less mode and a More mode. The user can toggle between these modes to display more or fewer controls or list columns.

There is a More/Less property (or a similar name) in the Applet Web Template Item object definition for each control, providing the ability to specify whether the control always appears, or only in More mode.

To specify that a control only shows up in the More mode a user would highlight it from the Applet layout editor, right mouse click and select “More” from the context menu.

NOTE: The More/Less feature is not supported for Pop-up applets. For more information about Pop-up applets, see [“Pop-Up Windows” on page 772](#).

Image Support

This section describes how to do the following:

- Configure images as bitmap objects
- Use image formats
- Use icons for field values

- Use images as links in controls and list columns
- Use images in template files
- Use image caching file manager

Configuring Images as Bitmap Objects

The Bitmap object in Tools supports image formats used in Web Applications like JPEG and GIF. The Bitmap object has an attribute called “File Name.” To configure a gif or a jpeg image in Tools, create a new Bitmap object and set its “File Name” attribute to the file name of the image file. When setting the image name assume that the images will be published in a folder named “images” under the public folder of the application. For images that are published within subfolders in the image folder, include the subfolder in the image name.

Example:

- For the image asterix.gif that is published in the eapps/public_enu/images folder, set the “File Name” attribute to “asterix.gif”.
- For the image next_on.gif that is published in the eapps/public_enu/images/bttns folder, set the “File Name” attribute to “bttns/next_on.gif”.

SWE will render the bitmap object using the HTML `` tag. The “Height” and “Width” attributes of the Bitmap object can be set to the height and width of the image that you want to display on the Web page. If these attributes are set, SWE will use them as “width” and “height” attributes of the `` tag. This allows the creation of various bitmap objects that share the same image file, but are rendered with different dimensions. The Bitmap object has another new attribute called “Alt Text.” This attribute can be set to the text to be used in the “alt” attribute of the image tag. The other attributes of the Bitmap Object like “Data” and “Transparent Color” are not used with Web Images.

SWE supports the use of these bitmap objects in various places like:

- In templates (as a replacement to the `` tags in the templates)
- As links for controls/list columns that invoke methods
- In the creation of the tree controls and hierarchical list applets

- In displaying icons for field values

SWE also has a feature where the images are published from the Siebel Server to the Web servers. Instead of being installed on the Web server, the image files are installed on the Siebel Server like the template files. These files are then automatically published to all the Web servers that connect to the Siebel Server.

You can configure your Bitmap objects so that the images are published to the `public_language/images` folder at runtime

where

language = the Siebel code for the Language Pack you have installed.

See [“Associating Images With Siebel Objects” on page 582](#) for more information about using the Bitmap object type.

Image Formats

GIF and JPG formats commonly used in Web clients. These types of images are not stored in the repository but instead are stored in files and referred to from the repository Bitmap objects.

- If the image format is not BMP, such as GIF, the image file name should be specified in File Name field. The corresponding image file should be located in the `Public\$lang\images` folder.
- After the File Name value is set, if you right-click on an entry in the Bitmaps applet in the Object List applet, the menus for Import Bitmap, Export Bitmap and Transparent Color are disabled since there is no need to import/export these bitmaps into the repository and since the concept of transparent color is not applicable. Files need to be copied to the Web server or Web client.
- If the image type is BMP, after the image is imported into the repository, the File Name field becomes read-only.

Using Icons for Field Values

SWE supports a feature to render control/list column field values as icons. A repository object called “Icon Map” supports this. Each icon map object is a collection of objects called “Icons.” An icon uses a bitmap object to define the image for the icon. Each icon within an icon map corresponds to a field value. Controls/List Columns have an attribute called “Icon Map” that can be set to the icon map object to be used for rendering the field values. Configuring and using these objects is explained below in detail.

For example, consider rendering the “Status” list column on the “Activity List Applet” using icons. The status field can have values “Not Started,” “In Progress,” and “Done.” Other values include `notstarted.gif`, `inprogress.gif`, and `done.gif` as the corresponding iconic image files.

NOTE: If you want to use custom icons in a list applet, you must size them in accordance with the list applet’s row font size. For example, when using an 8-pt font (standard for Siebel eBusiness Applications), icons should be 23 pixels in width x 14 pixels in height.

If you change the list applet row font size dynamically or place an icon larger than 23 x 14 in a row, the list applet rows will be scrambled.

To render a field using image files

- 1 Create a new Bitmap Category, for example, “Activity Status Icons,” to contain the bitmaps used for these icons.

Within this category, create bitmaps “NOTSTARTED”, “INPROGRESS” and “DONE.” These bitmaps should be configured to use the Web images notstarted.gif, inprogress.gif, and done.gif.

The Bitmap object in Tools has been extended to support image formats used in Web Applications like jpeg and gif. The Bitmap object has a new attribute called “File Name.” To configure a gif or a jpeg image in Tools, create a new Bitmap object and set its “File Name” attribute to the file name of the image file. When setting the image name assume that the images will be published in a folder named “images” under the public folder of the application. For images that are published within subfolders in the image folder, include the subfolder in the image name.

For example:

- For the image asterix.gif that is published in the eapps/public_enu/images folder, set the “File Name” attribute to “asterix.gif”.
- For the image next_on.gif that is published in the eapps/public_enu/images/bttns folder, set the “File Name” attribute to “bttns/next_on.gif”.

SWE will render the bitmap object using the HTML tag. The “Height” and “Width” attributes of the Bitmap object can be set to the height and width of the image that you want to display on the Web page. If these attributes are set, SWE will use them as “width” and “height” attributes of the tag. This will allow creation of various bitmap objects that share the same image file, but are rendered with different dimensions. The Bitmap object has another new attribute called “Alt Text”. This attribute can be set to the text to be used in the “alt” attribute of the image tag. The other attributes of the Bitmap Object like “Data” and “Transparent Color” are not used with Web Images.

SWE will support the use of these bitmap objects in various places like:

- In templates (as a replacement to the tags in the templates)
- As links for controls/list columns that invoke methods
- To create the tree controls and hierarchical list applets

- To show icons for field values
 - Other future uses
- 2 Create a new “Icon Map” object named, for example, “Activity Status.”
 - 3 Create “Icon” objects as child objects of the “Activity Status” icon map. Create one icon object for each field value that the status field can have. Set the “Name” attribute of the icon as the field value. Set the “Bitmap Category” and “Bitmap” attributes of the icon to the bitmap you want to show for the field value. Thus you will create the following icons in this example:

Name	Bitmap Category	Bitmap
Not Started	Activity Status Icons	NOT STARTED
In Progress	Activity Status Icons	INPROGRESS
Done	Activity Status Icons	DONE

- 4 Set the “Icon Map” attribute of the “Status” list column of the “Activity List Applet” to the icon map “Activity Status Icons.”

The procedure is the same with control field values.

SWE renders the image corresponding to the bitmap when the field value matches one of the icons defined.

If you create an icon named “Default” in a Icon Map object in tools and if the field value does not match any of the icons, then the icon named “Default” is used for the field. This feature is used to create an icon used with fields that could contain different values (for example, URLs). In this case, you would still set the HTML Type property of the field to be “URL” and its IconMap property to an IconMap object that contains only one icon named “Default.” If the field value does not match any of the icons and a “Default” icon is not defined, then the field value itself is rendered.

For more information about Icon Maps, see [“Icon Maps” on page 524](#).

Using Images as Links in Controls

You can use an image as a link with a control that invokes a method by using two attributes of controls called “HTML Bitmap” and “HTML Disabled Bitmap.”

The “HTML Bitmap” attribute is used to set the name of the bitmap to be rendered when the control is in an enabled state (that is, the method can be invoked), and the “HTML Disabled Bitmap” is used to set the name of the bitmap to be rendered when the control is in a disabled state.

SWE will use the “HTML Bitmap”/“HTML Disabled Bitmap” attributes only when the “HTML Type” of the control/list column is set to “Link.” If set to “Button,” the caption property of the control is used as the button label. You can use the “HTML Bitmap”/“HTML Disabled Bitmap” properties with custom HTML types. If you use the tag `<swe:this property="Data" type="Link"/>` within the definition of the custom html type in the SWF file, then SWE uses the bitmaps.

If the “HTML Bitmap”/“HTML Disabled Bitmap” attributes are not set, SWE will default to using the “Caption” property for the link.

These bitmaps have to be created in Tools under the Bitmap Category “HTML Control Icons.”

Image Caching File Manager

In eapps.cfg, set the following parameters:

```
[ImageCache]

swe_image_cache_root = d:/swe_cache/

swe_image_cache_num_subdirs = 256

swe_image_cache_size = 8092 //In KB

swe_image_cache_cleanup_idle = 7200 //in seconds

swe_image_cache_cleanup_threshold = 2880 //in minutes

swe_image_cache_enabled = TRUE //Default is True
```

The default values are set as above; one can change them to different ones.

The image caching file manager only caches the product catalog images in Web server. In order to test it, one has to set images in the product catalog view (ThumbNailImage field).

Hierarchical List Applets

The Siebel Web Engine supports rendering hierarchical list applets. These applets are used to show records that have a hierarchical relationship. These applets are modeled as list applets, and the hierarchy is implemented in the business component by setting the Hierarchy Parent Field attribute. These applets can be shown as a hierarchy in the Base and Edit List modes.

Hierarchical list applets do not need new templates. The icons required to display the hierarchy are added automatically to the first column in the list applet that is mapped to a business component field (Figure 233).

Figure 233. Hierarchical List Applet

It is also possible to show a hierarchical list applet using a Windows Explorer–like UI (very similar to the tree control). An example of this is the Categories list used to create and manage catalog categories in Siebel eSales.

The icons used for rendering the list applet should be specified in the repository using the HTML Hierarchy Bitmap object named in the HTML Hierarchy Bitmap attribute of the list object. The following bitmaps need to be specified for the HTML Hierarchy Bitmap—Expand Bitmap, Collapse Bitmap, and Space. The other bitmaps (except Arrow Down/Up Bitmap which is used only by tree controls) can be specified to get an applet that resembles the tree applet (which is not specified for hierarchical list applets in the current UI standards).

The hierarchical list applets also let you edit the hierarchy. This is done by adding controls that invoke the following methods on the applet (Table 58).

Table 58. Hierarchical List Applet Methods

Method	Action
Indent	Moves the current record to be a child of its preceding peer record (such as the record above)
Outdent	Moves the current record to be a peer of its parent record
MoveUp	Moves the current record up over its peer record
MoveDown	Moves the current record down over its peer record

When Indent and Outdent methods are invoked on a record, its relation to its child records does not change. The child records are also indented or outdented.

The MoveUp and MoveDown methods are used to position a record to invoke the Indent method. The changes made by the MoveUp and MoveDown methods are temporary; they are not saved to the database.

For a better appearance, the number of columns displayed in a hierarchical list applet should be small since the width of the column with the expand/collapse controls expand as a user navigates down the hierarchy. Similarly, the column with the expand/collapse controls should be a column with short field values.

About Grid Layout Templates

Grid Layout applet Web templates, Siebel tags, and enhanced features in the Web Layout Editor allow you to modify form layout without having to directly modify the underlying applet Web templates.

Standard applet Web templates (not grid-based) use placeholder tags to define an applet's layout. You could use the Web Layout Editor in Siebel Tools to map controls to any of the available placeholders, but you cannot use Web Layout Editor to change the layout of the placeholders themselves. To change the layout of the placeholders in these templates, you have to directly modify the applet Web template file.

Grid Layout applet Web templates use a pair of Siebel tags (`<swe:form-applet-layout>` and `</swe:form-applet-layout>`) that do not use placeholder tags. Instead they serve as a single container for all controls in the main body of a form applet. These tags enable you to use the Web Layout Editor as a WYSIWYG (what you see is what you get) design tool to configure the layout of form applets. In fact, this is the only way you can configure the layout of an applet based on a grid-based applet Web template.

The Grid Layout Web templates are:

- **CCAppletFormGridLayout.** This template may be used for form applets. See [Figure 234](#).

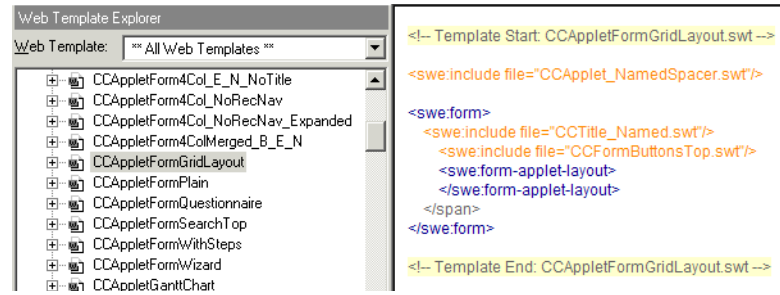


Figure 234. Grid-Based Applet Web Template for Form Applets

- **CCAppletPopupFormGridLayout.** This template may be used for popup form applets. See [Figure 235](#).

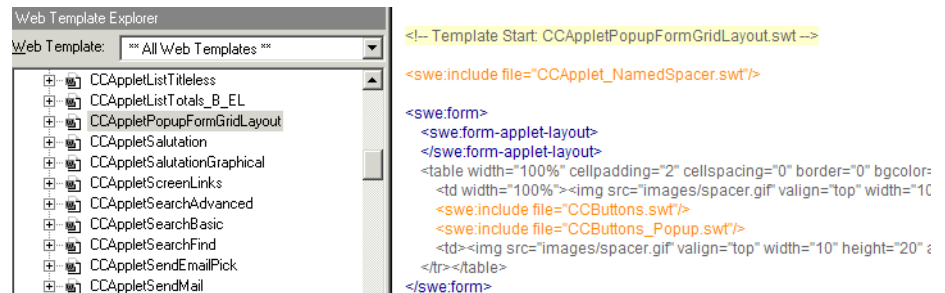


Figure 235. Grid-Based Applet Web Template for Popup Form Applets

Grid Layout applet Web templates consist of a body region and a header and footer region. The body region of the template is defined by the `swe:form-applet-layout` tag and contains no placeholder tags. However, the header and footer regions do use placeholder tags for buttons such as New and Save. You cannot edit the layout of header and footer regions using the grid layout features of the Web Layout editor.

The following list summarizes how grid-based applet Web templates differ from standard (not grid-based) applet Web templates:

- With grid-based templates, you can modify the layout of the form using Siebel Tools without having to modify the web template itself.
- With grid-based templates, Labels and Controls behave as separate items in the Web Layout Editor. This allows you to place them independently in the applet layout. However, Labels and Controls are really a single object in the repository with one set of shared properties.
- Grid-based templates do not automatically compress empty space in a column.

You can modify the background colors of applets based on Grid Layout Web templates by modifying the appropriate selectors in the cascading style sheet, `main.css`.

- When the applet is the parent applet (the top applet on a view) modify the `.AppletStyle1` selector as shown in the example below:

```
/*Parent Applet Style*/  
  
.AppletStyle1{background-color : #f00000; color:#00f0ff; }
```

- When the applet is the child applet (not the top applet on the view) modify `.AppletStyle3` selector as shown in the example below:

```
/*Child Applet Style*/  
  
.AppletStyle3 {background-color : #f0f000; }
```

For more information about modifying `main.css`, see *Developing and Deploying Siebel eBusiness Applications*.

Grid-based templates make configuring the layout of form applets much simpler than configuring the layout of applets based on standard (not grid-based) templates. Using Grid Layout templates makes the process of designing a form applet similar to working in other palette-based graphics design applications.

See [“About Grid Layout” on page 547](#) for more information about Grid Layout.

Creating Custom HTML Control Types

Siebel applications support the use of several different control types out of the box (for example, Check Box, Button, Mail To, Text Area, and so on). However, there may be cases in which additional HTML types are needed. You can define your own HTML types by adding type definitions to the .SWF file.

Unlike cascading style sheets, which are used primarily to define general stylistic information about labels, titles, background colors, and so on, user-defined types in the CCHTMLTYPE.SWF file would normally be used to define more complex attributes that determine either the appearance or client-side functionality of a type of HTML element. Examples might include a button type that is associated with a particular GIF or a type of link that connects the user with an FTP site.

Again, these attributes could also be given to a page element by specifying the appropriate tags and attributes directly in the .SWT file. However, by defining them as types within the CCHTMLTYPE.SWF file, they can be referenced from within Siebel Tools for a specific control on a specific Applet Web Template or Web Page object. This preserves the generality of .SWT templates by avoiding the need to place HTML directly within them. It also improves the maintainability of the application by minimizing customization done to templates and storing more configuration information within the repository.

To create and use a new HTML Type

- 1** Add the name of the new type (for example, “MiniButton”) to the List of Values used for the HTML Type property in Siebel Tools (REPOSITORY_HTML_CTRL_TYPE).
- 2** Add the formatting information for the new type to one of the application's two .SWF files. These files, which should use the extension .SWF, must reside in the same directory as the template files. One file contains the special types defined by Siebel Systems, Inc.; the other contains customer definitions, either to add additional types or to override Siebel types.

The format for rendering the custom type is specified by using two tags:

- `<swe:htmltype>`

- `<swe:this>`

The details of defining a new format are explained below, but for now, the form is as follows:

```
<swe:htmltype name="XXX" mode="AAA" state="BBB">
    . . . . . HTML . . . . .
<swe:this property="YYY" />
    . . . . More HTML . . . .
</swe:htmltype>
```

- 3 Specify the names of the .SWF files to be used by an application in the [SWE] section of the .CFG file used by the application's object manager.

```
[SWE]
SystemSWFName = CCHtmlType.swf
UserSWFName = MyFormat.swf // You must set this name.
```

- 4 In Siebel Tools, change the HTML Type property of the control, list column, or page item to the new type.
- 5 In the template file, use the FormattedHTML property for the `<swe:control>` or `<swe:pageItem>` element.

When SWE Uses a Custom HTML Type

If the HTML type of a control, list column, or page item is a custom type, the Siebel Web Engine will use the .SWF format when rendering any elements that are mapped to the control, and that specify the FormattedHtml property.

The following cases exist:

- `<swe:control id="1" property="FormattedHtml" />`
- `<swe:control id="1"> ... <swe:this property="FormattedHtml" /> ... </swe:control>`
- `<swe:pageitem id="1" property="FormattedHtml" />`

```
■ <swe:pageitem id="1"> ... <swe:this property="FormattedHtml"/>
   ... </swe:pageitem>
```

The formatting will not be used with any other property, such as Display Name. However, in the .SWF file, the `<swe:this>` element can refer to these properties (except the FormattedHtml property itself).

More About Format

The .SWF files contain multiple format specs. Each format spec includes two parts:

- An enclosing XML element that names the type and optionally names the mode and state in which the current format is used
- The enclosed format content

The format content syntax follows these rules:

- It must be valid, regular SWE syntax.
- It may refer to all the properties of the current control, except “FormattedHtml” (in order to prevent recursion).
- It may use a new `swe:this` property, “Data,” which is explained below.

Examples

Example 1: To create a new HTML type for a control called “LabelRed” that shows the caption of the control in red, the formatting might look like this.

```
<swe:htmltype name="LabelRed">
    <font color="red"> <swe:this property="DisplayName"/> </font>
</swe:htmltype>
```

Example 2: The Data property of `<swe:this>` is something like a macro that casts the current, custom type to one of the intrinsic types, and then inserts the FormattedHtml property of the intrinsic type. To use the Data property, you add an additional Type attribute to the `<swe:this>` tag, that names the intrinsic type. For example, to create a new type called “MiniButton,” in which special formatting is added to the Web Engine intrinsic type Link, you might write a format something like this:

```
<swe:htmltype name="MiniButton">
```

```
<img SRC="images/btn_left.gif" border=0 height=15 width=2>
<swe:this property="Data" type="Link" />

</swe:htmltype>
```

Here the `<swe:this property="Data" type="Link" />` will output the same HTML as if the template included a separate `<swe:this>` tag, where the property was `FormattedHtml`, and the HTML type of the control were the built-in type `Link`.

You can only specify built-in types and not custom types for the `type` attribute of `Data` elements.

Example 3: You can also define custom formats for the different applet modes (Base, Edit, New, Query), by using the `Mode` attribute for the `<swe:htmltype>` tag. If a mode is specified, then that formatting is used only if the current show mode matches the value specified for this attribute. For example, if you want to create a new HTML type called `SiebelText` to show a control which displays as a label and a text field while in Edit template, and as read-only text in Base mode, you specify the format as:

```
<swe:htmltype name="SiebelText">
  <swe:this property="Data" type="Text"/>
</swe:htmltype>
<swe:htmltype name="SiebelText" mode="Edit">
  <swe:this property="DisplayName"/>:&nbsp;<swe:this
  property="Data" type="Text"/>
</swe:htmltype>
```

Example 4: You can define another optional attribute to the `<swe:htmltype>` tag, namely `State`, to show different formatting depending on the state of the control or list item. Currently Siebel Web Engine supports two states:

- **Disabled.** For controls or list columns that invoke methods, where the method cannot be invoked on the record.
- **Required.** For controls or list columns that are required.

For example, to show grayed-out buttons when a method cannot be invoked, add the following format definition in addition to the default definition shown earlier.

```
<swe:htmltype name="MiniButton" state="Disabled">
    
    <swe:this property="Data" type="Link" />
    
</swe:htmltype>
```

With built-in HTML types, if a method cannot be invoked, then the control or list item is not shown (same as the current behavior). With custom HTML types, however, the formatting specified in the .SWF file is always shown. The HTML generated for the property Data (`<swe:this property="Data" type="Link" />`) when the method cannot be invoked is simply the caption of the control or list item without any `<a href>` tags.

You can hide a control or list item with a custom HTML type when a method cannot be invoked by creating an empty `<swe:htmltype>` tag for the Disabled state.

```
<swe:htmltype name="MiniButton" state="Disabled"></swe:htmltype>
```

NOTE: This only hides the `<swe:control>` or `<swe:this>` tag that invokes the `FormattedHtml` property.

Example 5: To show the `SiebelText` type with an indicator (*) for required fields you can add the following format definition in addition to the definitions for this type shown earlier.

```
<swe:htmltype name="SiebelText" mode="Edit" state="Required">
    *&nbsp;
    <swe:this property="DisplayName" />
    :&nbsp;
    <swe:this property="Data" type="Text" />
```

```
</swe:html type>
```

NOTE: When SWE looks up HTML Type definitions in the .SWF file, the order of precedence will be Mode and then State. It is recommended to always create a default format definition (that is, without specifying the mode and state attributes) for all custom HTML types.

Removing HTML Frames From Web Templates

Default Web templates are designed to use HTML frames to display Siebel applications in distinct regions of a page, such as the menu bar, screen tabs, and content area. In most cases, using frames is desirable. However, for some partner and customer applications, such as eSales or PRM, there may be cases where frames are undesirable. For example, a company may have corporate style guidelines that restrict the use of frames and require Web pages to be displayed in a single, scrolling region.

You can configure customer and partner applications to run without visible frames. You do this by modifying several of the default Web templates. Be aware that there are certain features that rely on frames and therefore will not be available if HTML frames are removed. These known issues are described below in [“Known Issues When Running Siebel Applications Without HTML Frames.”](#)

NOTE: Employee applications, such as Siebel Sales or Siebel Call Center, require HTML frames and cannot be configured to run without them.

To run customer and partner applications without visible HTML frames, you need to modify the following Web templates:

- page container
- page header and footer
- views with custom headers and footers

Modifying Page Containers

The default template for page containers consists of an HTML frameset that defines the main regions of the page. You can remove the frameset and insert the referred to HTML directly into the page container.

Modifying Headers and Footers

After you have modified the container page so that it runs without frames, you must modify the view containers so that they work in the new page container. By default, view containers contain `<head>` and `<body>` tags because they are designed to run in frames. However, when running without frames these view containers are inserted inline into the HTML page. To operate correctly this requires that the view container not insert these extraneous `<head>` and `<body>` tags.

Most views include standard header and footer Web template files which are responsible for adding the `<head>` and `<body>` tags. By modifying these files, you can enable most views to operate correctly in frameless mode.

To update all customer views for running without frames, replace the contents of the header and footer Web templates with a single placeholder `` tag. Note that while these modified Web templates are in use, the views that use them will no longer operate correctly when used with frames.

Partner applications (PRM) use a number of employee views. Therefore you must also update employee headers and footers. When the modified Web template files are applied, any employee applications using them will not operate correctly. It is recommended that frameless PRM and employee applications use distinct Web-template directories.

Modifying Views with Custom Headers and Footers

There are a small number of special-purpose views that do not use the standard header and footer .swt files. To enable these views to run in a frameless mode, delete the `<head>` block and the `<body>` and `</body>` tags so that only the contents that were contained within the body block remain.

Known Issues When Running Siebel Applications Without HTML Frames

Certain features in Siebel applications are designed to work with HTML frames. When you remove frames, these features will not behave as expected. The following are known issues that occur when you run customer or partner applications without HTML frames.

- **Top Level Hidden Frames.**
 - While the approach described in this section eliminates any visible frames in customer and partner applications, Siebel applications still generate a top level frameset and one or more hidden frames.
- **Partner Relationship Manager.**
 - Search Center and Customer Dashboard are areas of the UI that can be turned on or off by a user. These are implemented using frames, and are not available in frameless mode.
 - InfoCenter Explorer view is not displayed correctly in frameless mode.
 - Calender view is not displayed correctly in frameless mode.
- **eCustomer.**
 - Clicking the Go button or the Advanced Search link opens a new browser window. This is due to Search Center not being available in frameless mode.
 - Parametric Search view does not work correctly in frameless mode.
- **eConfigurator.**
 - The eConfigurator runtime does not operate correctly in frameless mode.
- **eAdvisor.**
 - eAdvisor is not supported in frameless mode.

Template Configuration Features

The following section describes the template configuration features.

Displaying Server Side Errors

When a server side error occurs on submitting a form, SWE shows the same page again with the error message displayed within the page. For errors that occur outside of a form submission SWE continues to use the application's Error Page.

This error message display is mainly developed for showing error messages within a form. It is also used to show an error message in an error page to replace the to be deprecated `pageitem.errorMsg` way of showing error messages.

To display the error message within a form, place the following tags inside `<swe:form>` tag:

```
<swe:error>
    <swe:this property="FormattedHtml" />
</swe:error>
```

The error messages are shown in plain text, but each error message takes a new paragraph. It is the responsibility of the enclosing HTML tags to modify the font and style of the error message. Sometimes, the error message may not be visible; this is because the font uses the same color as the background.

If the application developer does not use error tags in the swt files, the code automatically generates an error node (a `CSSSWEErrorSWX` instance). This automatically generated error node is inserted as the first child of the enclosing page/form node.

The syntax of the `<swe:error>` tag is as follows:

■ `<swe:error>`

Usage:

```
<swe:error property="FormattedHtml" />
```

or

```
<swe:error>
    <swe:this property="FormattedHtml" />
```

Special Behavior Supported by Templates

Template Configuration Features

```
<swe:error/>
```

This tag should be used within all `<swe:form>` tags.

You should also use this tag instead of the `<swe:pageitem>` tag mapped to the “_SWEErrMsg” item in the application's Error Page. The use of the “_SWEErrMsg” item is deprecated for 7.0.

An example of the use of this tag is:

```
<swe:form>
  <swe:error>
    <b><font color="red"> <swe:this property="FormattedHtml"/> </font></b>
  </swe:error>
  ...
</swe:form>
```

When the form is being rendered when there are no errors, the contents of the `<swe:error>` tag will be skipped.

Adding Graphics

To enhance the appearance or navigation of your application, you can create .GIF files and include links to them from the HTML pages.

Creating Directories for Your Graphics Files

Your Siebel Web installation includes three directories for your application. These directories will contain all the files used by your application, including the graphics files.

You should place your graphics in the `public\lang\images` directory. This directory gets created during the installation of the Siebel Web applications.

Adding Sorting Capabilities to Your Application

To add sorting capabilities to your applications, you use the template file for the Sort Dialog by specifying the Application property Sort Web Page. You can use the Sort Dialog to show one or more instances of a list of fields to sort on, and the sorting order to use. You can invoke the Sort Dialog using a Control that invokes the `SortOrder` method. This control should be used only in base templates of List Applets.

You use the tag `<swe:sort-field>` to show the list of sortable fields and the sorting order options. This tag takes one attribute called `sequence`, which specifies the sort column order. This is a required attribute.

```
<swe:sort-field sequence="1"/>
```

This tag will render two HTML select lists. The first select list will show the list of fields that can be sorted and have been mapped to `<swe:control>` tags in the base template for the applet. The second select list will show the two options for sorting order: Ascending and Descending. You can have as many `<swe:sort-field>` tags in the Sort Web Page as you want. Each `<swe:sort-field>` tag should specify the order in which the selected columns should be sorted, using the `sequence` attribute.

To create the link or button that would execute the sort, create a Web Page Item that invokes the `ExecuteSort` method. You do not have to specify the parameters View and Applet for this method; these will default to the currently active view and applet.

Example

Below is a fragment from a sample Sort Web Page.

```
<swe:form>

<table width=100% bgcolor="#EEEEEE" border=0 cellspacing=0
cellpadding=3>

<tr>

  <td><swe:pageitem id="1" property="DisplayName"/></td>

  <!--"Sort By" Label -->

</tr>

<tr>

  <td><swe:sort-field sequence="1"/></td>

  <!-- First column to sort on -->

</tr>

<tr>

  <td><swe:pageitem id="2" property="DisplayName"/></td>

  <!-- "Then By" Label -->

</tr>

<tr>

  <td><swe:sort-field sequence="2" /></td>

  <!-- Second column to sort on -->

</tr>

<tr>

  <td><swe:pageitem id="2" property="DisplayName"/> </td>

  <!-- "Then By" Label -->

</tr>

<tr>
```

```
<td><swe:sort-field sequence="3"/></td>
<!-- Third column to sort on --></tr>
<tr>
  <td><swe:pageitem id="5" property="FormattedHtml"/></td>
  <!-- Execute Sort -->
</tr>
</table>
</swe:form>
```

Cascading Style Sheets

The look and feel of user interface elements are controlled by cascading style sheets. Cascading style sheets contain classes that define elements such as color schemes and fonts. Cascading style sheet files (.css files) are located in:

- The Siebel Server installation directory

siebsrvr_root\WEBMASTER\files\language_code

- The Mobile or Dedicated Web Client installation directory

client_root\PUBLIC\language_code\FILES.

- The Tools installation directory

tools_root\PUBLIC\language_code\FILES

The following are examples of how you could use cascading style sheets to modify the look and feel of the user interface:

- Have text appear in the font of your choice
- Specify that size of text in points, pixels, and many other units
- Add any color or background color for images

The .SWT templates can be configured to use formatting tags. By storing style-related information in cascading style sheets rather than .SWT templates, you can increase the modularity and consistency of your applications and the ease with which the .SWT templates can be modified and reused.

See *Siebel Developer's Reference* for more information about cascading style sheets.

Since style-related information stored in cascading style sheets is rendered slightly differently in different browsers, customers should test the results in both browsers unless their users are restricted to one or the other.

See *Siebel Developer's Reference* for cascading style sheet definitions.

This chapter explains how to:

- Archive, export, and import entire projects and individual repository objects
- Rename, delete, back up, and migrate repositories
- Create patches
- Manage your Siebel repositories with third-party source control software

Code Pages and Unicode

Siebel applications support migrating repository data from the source environments to the target environments listed in [Table 59](#). This applies to importing, exporting, backing up, or migrating repository data using any of the methods described in this chapter.

Table 59. Code Pages and Unicode Support for Repository Migration

Source Environment	Target Environment
Code page	Code page
Unicode	Unicode
Unicode	Code page
Code Page	Unicode

Exporting and Importing Repository Objects

Archive files contain object definitions exported from a repository; they are available for importing into other repositories.

Archive files:

- Have an .sif extension, by default
- Can be version-controlled using third-party source control systems
- Are used to share object definitions among repositories in a multiple repository development effort
- Can only be exported and imported into repositories with the same repository schema definition

You can include any of the following in an export file:

- One object definition
- Multiple object definitions of various object types
- All object definitions assigned to a specific project or multiple projects

NOTE: The Project property is never exported to or imported from an archive file.

When you do the import you can specify conflict resolution at the object-definition level, telling the system whether to ignore the imported object definition, replace an existing definition with an imported one, or merge the two on a property-by-property basis.

CAUTION: Exported objects from one version of Siebel eBusiness Applications should not be imported (through .sif files) into a different version, as object definitions might have changed. Importing an invalid object might result in invalid configurations and an unsupported application.

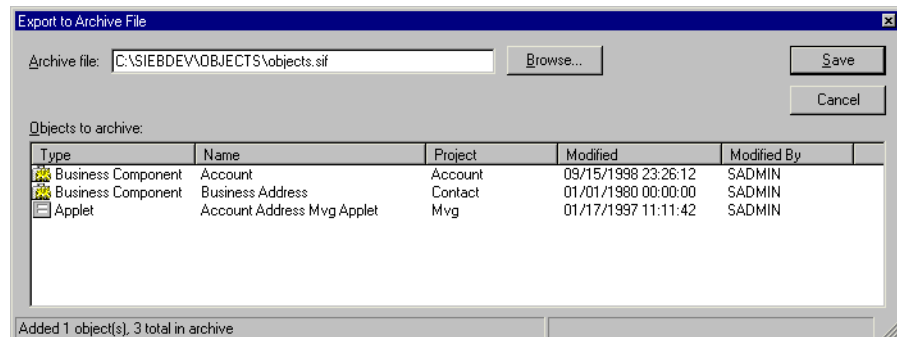
Exporting Individual Object Definitions

Only top-level object types—the object types that are visible in the Object List Editor when you click an object type in the Types tab of the Object Explorer—can be exported. Child object definitions are exported and imported along with their parents.

To export individual object definitions

- 1 In the Object Explorer, navigate to the object type of the object you want to export.
- 2 In the Object List Editor, select the object or objects you want to archive.
- 3 Select Tools > Add To Archive from the menu bar.

The Export to Archive File dialog box appears.



Status messages appear showing which child object definitions are being included. When the process completes, the selected top-level object definition or definitions appear in the Objects to Archive list table in the Export to Archive File dialog box.

- 4 If you need to add object definitions of another object type, navigate to that object type in the Object Explorer without closing or saving the Export to Archive File dialog box.

Move the Export to Archive File dialog box out of the way, if necessary.

- 5 Repeat [Step 2](#) through [Step 4](#) for each object you want to archive.

Repositories

Exporting and Importing Repository Objects

- 6 If you want to remove an object definition from the list, select it and press DEL.
- 7 When you are finished adding object definitions to the list, enter in the Archive File box the path and filename of the archive file to create. Alternatively, you can browse to an existing archive file using the Browse button and the resulting Save As dialog box.
- 8 Click Save. If a file of this name and location already exists, you are prompted to overwrite it or cancel.

Figure 236 shows the Account Business Component ready to be exported.

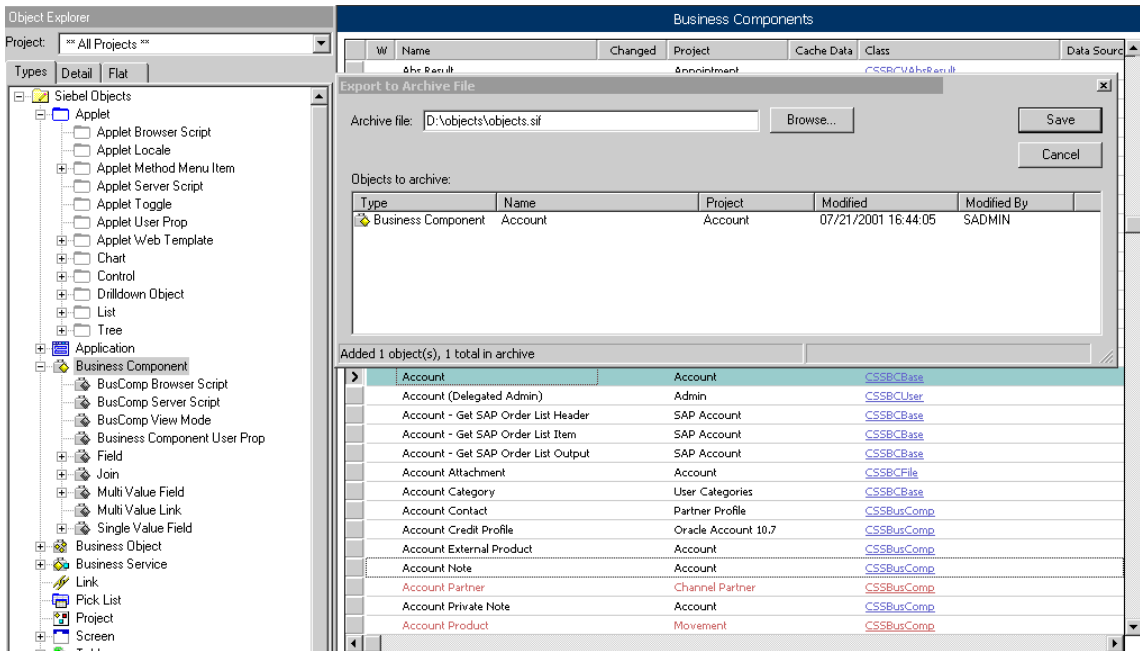


Figure 236. Exporting the Account Business Component

Exporting Object Definitions Using the Command Line Interface

You can also export object definitions using the Command Line Interface. The command line interface is invoked from the `siebdev` executable using the command switch `/batchexport`. The `siebdev.exe` is located in the Bin directory of the Siebel Tools installation directory.

The syntax of the `/batchexport` switch is:

```
siebdev.exe /c <config file> /d <database> /u <user name> /p  
<password> /batchexport <Repository Name> <Input File Name> <Log  
File>
```

The command line interface provided by the `/batchexport` switch accepts an input file that specifies export objects. The input file takes a comma-delimited format of *Object Type*, *Object Name Search Expression*, and *.sif file name*. The search expression takes any Tools accepted query criteria. To specify *.sif file*, you can use absolute file path or relative file path to the current directory.

You can place multiple lines in the input file, each requesting to export multiple objects into one *.sif file*. However, if you specify the same *.sif export file* in multiple lines, only the last export will take effect—the previous exports will be overwritten.

As an example, the following content in an input file would request the `batchexport` switch to export all business components whose name is like “*Account*” into exports.sif:

```
"Business Component,*Account*,export.sif"
```

NOTE: There should be no space before and after commas.

The following sample export command would export objects specified in the input file, `obj.txt`. It will also log results into `export.log`:

```
siebdev.exe /c tools.cfg /d sample /u sadmin /p sadmin /  
batchexport "siebel repository" obj.txt export.log
```

Exporting Entire Projects

The following procedure describes how to export an entire project.

To export an entire project or projects

- 1 In the Object Explorer, navigate to the Project object type.
- 2 In the Object List Editor, click the Project object definition to select it.
- 3 Select Tools > Add to Archive from the menu bar.

The Export to Archive File dialog box appears.

- 4 Enter the path and filename of the archive file to be created in the Archive File box.
- 5 Click Save.

Figure 237 illustrates this procedure.

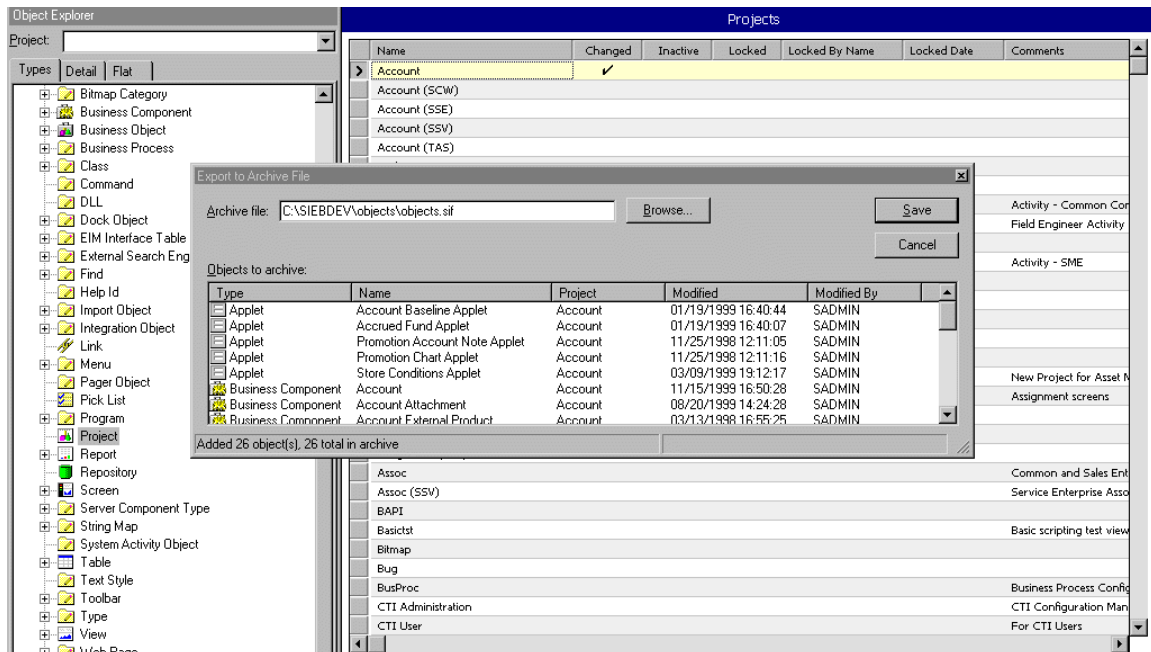


Figure 237. Exporting the Account Project

NOTE: It is possible, but inadvisable, to export an entire repository by adding a Repository object definition to an export list. The resulting export file will contain all object definitions in the repository, but the processes of exporting and importing such an archive will take an extremely long time, and the archive file will be very large. If you need to export an entire repository, refer to [“Renaming, Deleting, Backing Up, and Migrating Repositories”](#) on page 932.

Importing Object Definitions

You can import object definitions from an archive file into a local repository.

The Import process consists of the following major steps (described in the following sections):

- 1 Preparing the target repository for import
- 2 Loading an archive file into the Import Wizard’s preview window, and specifying the default resolution of conflicts between imported and pre-existing object definitions
- 3 Adjusting the details of conflict resolution
- 4 Importing as specified
- 5 Testing and checking in the changed projects

Preparing the Target Repository for Import

You need to import into a checked-out project or projects on the local database of a client computer—do not import to the Server database. Make sure the following conditions exist before importing:

- The import file is accessible to the local machine by way of the network or local drives.
- The target repository is open in Siebel Tools and is the active repository.

- The projects that will be affected by import have been checked out to the local database. This includes any project that any object definition in the export file is assigned to.

The only exception consists of projects (or their object definitions) that are in the archive file, but that do not exist yet in the target repository. These are not checked out because they do not exist in the target repository.

NOTE: In some cases it may be difficult to know in advance which projects need to be checked out. The Import wizard informs you of any projects that were not locked but need to be. This occurs on the second panel of the Import wizard, after it has analyzed the object definitions in the archive file and compared them to the object definitions in your repository.

Loading Object Definitions from an Archive File Into the Preview Window

You need to load the object definitions from an archive file into preview window to resolve any conflicts between objects.

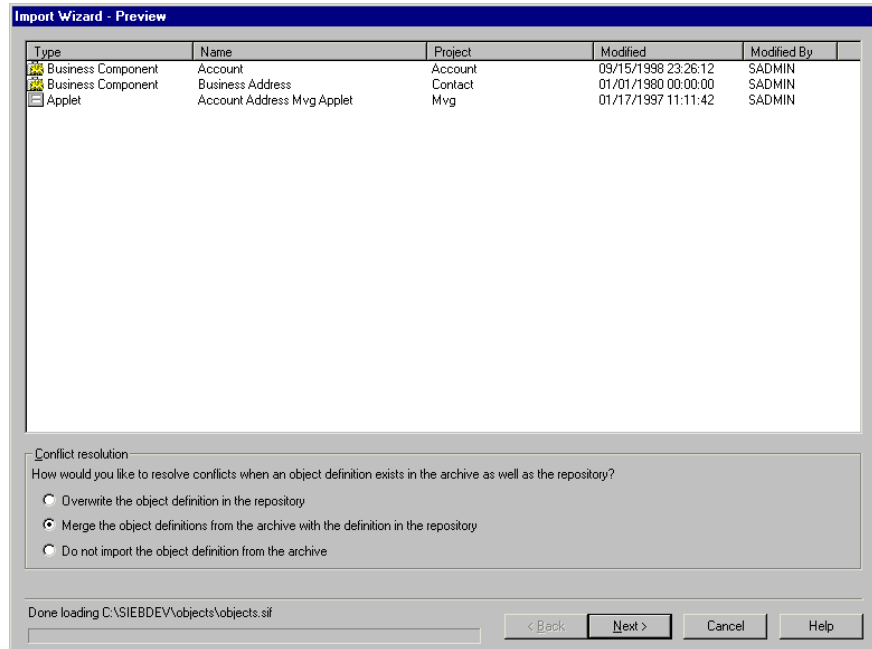
To load object definitions from an archive file into the preview window

- 1** Open the target repository in Siebel Tools, if it is not already open.
- 2** Select Tools > Import From Archive.

The Select Archive To Import dialog box appears.

- 3 Navigate to the archive (.sif) file in the dialog box and click Open.

The Import Wizard - Preview window appears.



This window identifies the projects and the nonproject top-level object definitions in the archive file you have opened so you can preview the contents of the archive file.

- 4 Select a radio button in the Conflict Resolution radio button group.

This specifies the default resolution for conflicts between the archive file and the target repository. You will have the opportunity in subsequent windows in the Import Wizard to change this choice for individual object definitions.

There are three choices available in the Conflict Resolution radio button group, as follows:

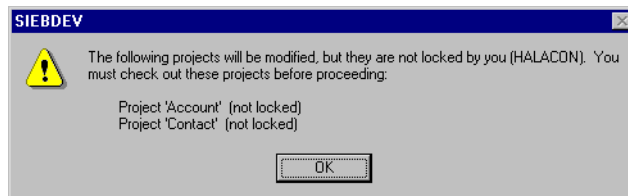
- **Overwrite the object definition in the repository.** If the same top-level object definition is found in the archive file and target repository, delete the version in the target repository, along with its children, and replace them with the object definition and children from the archive file.
- **Merge the object definitions from the archive with the definition in the repository.** Merging is the default, and generally the safest option. When the same top-level object definition occurs in both the target repository and the archive file:
 - Replace differing properties in the target top-level and child-level definitions with those in the file being imported.
 - Add new child object definitions to the target repository if they are not already present.
 - Do not change child object definitions in the target repository that are not also present in the archive file.

The resulting top-level object definition has the same properties and children as the object definition in the archive, plus any children that were already present in the repository definition.

- **Do not import the object definition from the archive.** Do not change the object definitions in the target repository.

5 Validate your selections and click Next to proceed.

6 If there are object definitions you will be replacing or modifying and whose projects are not locked, the following warning message appears:



If this message appears, you need to cancel the import process, lock the projects, and restart the Import Wizard.

Adjusting the Details of Conflict Resolution

You have made general decisions about conflict resolution in the Conflict Resolution radio button group in the previous window of the Import wizard. In the Review Conflicts and Actions window (shown in [Figure 238](#)), you can make adjustments for individual object definitions and properties.

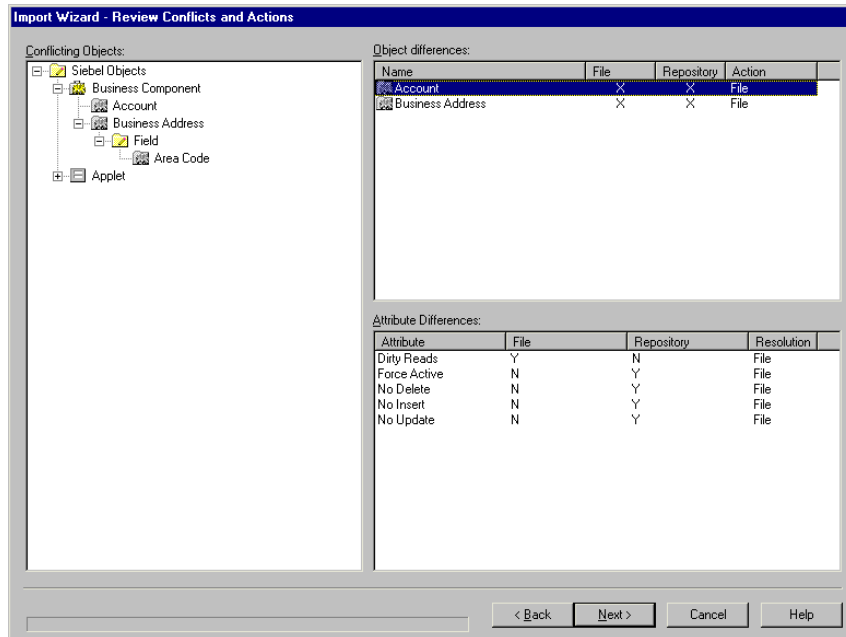


Figure 238. Review Conflicts and Actions Window

The three windowpanes are Conflicting Objects, Object Differences, and Attribute Differences.

Conflicting Objects Windowpane

The Conflicting Objects windowpane displays the hierarchy of object definitions for which there are differences. It is used for navigation and behaves like the Object Explorer in Detail mode. The hierarchy in this windowpane mirrors the object type/object definition hierarchy in a Siebel repository, but shows only conflicts to resolve rather than all repository or archive object definitions.

If you select an object definition in the hierarchy, it and others at its level appear in the Object Differences windowpane.

Object Differences Windowpane

The Object Differences windowpane displays object definitions, one to a row. It shows for each object definition whether it exists only in the archive file, only in the target repository, or in both, and what resolution is specified. You can change the resolution here.

The object definitions displayed in the Object Differences windowpane include those at all hierarchical levels, not just top-level object definitions. This provides the means for making adjustments to the resolution for any affected object definitions.

The File and Repository list columns indicate whether each identified object definition is present in the archive file or target repository. An “X” indicating the object definition’s presence can appear in the File list column, the Repository list column, or both. These list columns are for information only; you cannot change the check marks.

The Action list column indicates the proposed resolution for each object definition in the list. This setting is initially generated for each object definition from the default behavior selected in the Conflict Resolution radio button group in the Preview window. You can right-click on the value in the Action list column and select a different value from a shortcut menu. The available selections include the following:

- **File.** Equivalent to the “Overwrite the object definition in the repository” selection in the previous window.
- **Merge.** Equivalent to the “Merge the object definitions from the archive with the definition in the repository” option in the previous window.

The resulting top-level object definition has the same properties and children as the object definition in the archive, plus any children that were present in the repository definition.

- **Repository.** Equivalent to the “Do not import the object definition from the archive” option in the previous window.

For more information about these options, see [“Loading Object Definitions from an Archive File Into the Preview Window”](#) on page 924.

Attribute Differences Windowpane

The Attribute Differences windowpane displays the property value conflicts for the currently selected object definition in the Object Differences windowpane. Those properties are listed only where there is a conflict.

The Attribute Differences list contains the following list columns:

- **Attribute List Column.** Name of the property.
- **File List Column.** Value of the property in the archive file version of the object definition.
- **Repository List Column.** Value of the property in the target-repository version of the object definition.
- **Resolution List Column.** Value of either File or Repository for each property, depending on whether the archive-file or target-repository version of the object definition is to determine the value of the property in the final definition.

This list column can be updated only if the object definition whose properties are being displayed has an Action setting of Merge in the Object Differences list. Otherwise, the shortcut menu options are read-only and are grayed out, and the value displayed is the same as that in the Action column of the Object Differences list.

To change the Resolution value from Repository to File or the reverse, right-click on the Attribute row to change and select Repository or File from the shortcut menu.

After you click Next in the Review Conflicts and Actions window, you are shown a summary of your changes and prompted to proceed or cancel, as shown in [Figure 239](#).

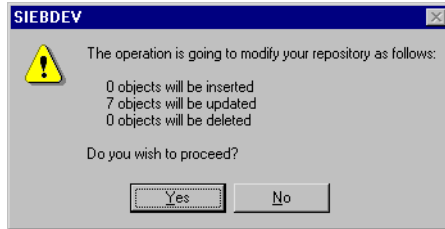


Figure 239. Summary Prompt

If you choose to proceed, the Summary window appears, as shown in [Figure 240](#).

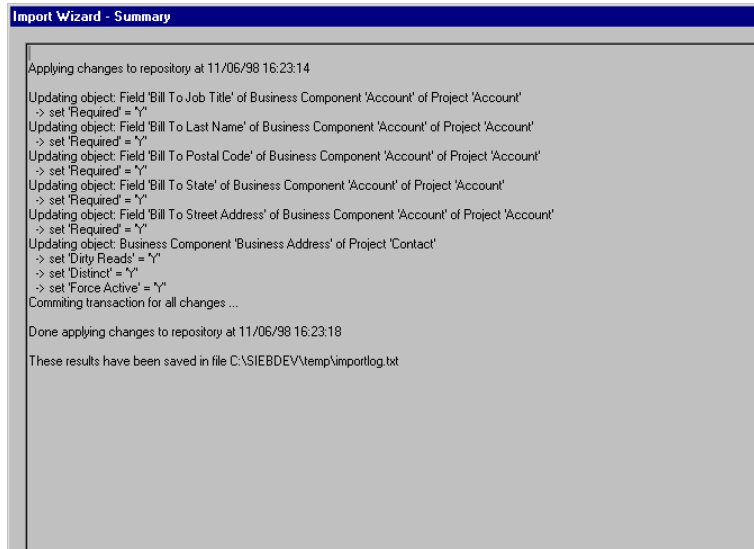


Figure 240. Summary Window

Importing occurs while the Summary window is displayed. The import actions are logged and displayed as they occur. When the import process is completed, click Finish.

A log file named importlog.txt is created in c:\sea7xx\tools \temp. It contains the same list of messages that appeared in the Summary window. You may find it useful to store this file elsewhere for a record of what changes were made to the repository. It is also a good idea to change the filename so it reflects the date of the import.

Importing Object Definitions Using the Command Line Interface

You can also import object definitions using the Command Line Interface. The command line interface is invoked from the siebdev executable using the command switch /batchimport. The siebdev.exe is located in the Bin directory of the Siebel Tools installation directory.

The syntax of the /batchimport switch is:

```
siebdev.exe /c <config file> /d <database> /u <user name> /p  
<password> /batchimport <Siebel Repository name> <Import Mode>  
<.sif file1, .sif file2, .sif fileN; or directory where .sif files  
can be found> <log file>
```

NOTE: You can specify .sif file and log file by full path or relative path to the current directory.

For example, the following sample import command imports import1.sif, located in the parent directory, and import2.sif, located in the Tools directory, into the Siebel repository using the overwrite mode. It also logs the results to import.log:

```
siebdev.exe /c tools.cfg /d sample /u sadmin /p sadmin /  
batchimport "siebel repository" overwrite ..\import1.sif  
c:\Tools\import2.sif import.log
```

The following sample import command imports all files under c:\tools\importfiledir into the Siebel repository using the merge mode. It also logs the results to import.log.

```
siebdev.exe /c tools.cfg /d sample /u sadmin /p sadmin /  
batchimport "siebel repository" merge c:\tools\importfiledir  
import.log
```

Renaming, Deleting, Backing Up, and Migrating Repositories

Various executable (.exe) programs are available outside Siebel Tools for working with an entire repository as a unit. These programs are found on the application server and database server. This section describes the use of these programs, as well as other procedures you can perform that act on an entire repository.

Development tools are installed with the Siebel Server to manage repositories across enterprises (development, test, and production) that do the following:

- Import a repository
- Export a repository
- Move a repository from a source environment to a target environment

All of these utilities manage the entire repository definition. To manage projects and objects within a given repository, the archive functionality in Tools is recommended.

CAUTION: If versions of these program files appear in your Siebdev directory or its subdirectories, do not use the ones found there. Use only the programs found on the server.

Renaming and Deleting Repositories

You might need to rename a repository in some situations. However, renaming can cause problems unless all developers check in their checked-out projects prior to renaming the repository. Following the renaming, they need to do a get on all projects. Siebel Systems recommends that the current active repository in production always be named Siebel Repository.

To rename a repository

NOTE: You should be connected directly to the server database when you do these steps.

- 1 Have all developers check in all projects that have been checked out from this repository.
- 2 In the Object Explorer, select the Repository object type.

NOTE: The type is not visible by default in the Object Explorer; it must be selected by choosing View > Options, selecting the Object Explorer tab, and moving Repositories into the “Visible top level objects” section of the Development Tools Options dialog box.

- 3 In the Object List Editor, click in the Name property of the repository you want to rename.
- 4 Enter the new name.
- 5 Click outside of the record to save your changes.
- 6 Have developers do a get of all projects.

In the Server Parameters view (accessed from the application-level menu by choosing View > Site Map > Server Administration > Servers > Server Parameters), verify references to the repository name in all Siebel client .cfg files and Siebel Servers. The name of the current production repository needs to be Siebel Repository. You should indicate which server parameter is identified by the Siebel Repository and show how it can be changed and also indicate the specific parameter name in the .cfg files that reference the repository name.

Deleting a Repository

The delete process remove all records associated with the repository. It is recommended that you only delete a repository after you have verified that you do not need any contents in this repository. It is best to export and archive the repository if you are unsure.

NOTE: To copy a repository, use the Import/Export Repository option.

To delete a repository

- 1 Click the Repository object type in the Object Explorer.

- 2 In the Object List Editor, click anywhere in the row for the repository you want to delete.
- 3 Choose Edit > Delete Record.
- 4 Click outside the record to commit the Delete action.

NOTE: Deleting a repository takes a long time and requires system resources such as rollback segment, cursors, tablespace, and so on. You might want to consult your DBA to check on system resources before deleting a repository.

Backing Up and Restoring Repositories

A configuration utility is generally used when backing up and restoring a repository. You use this utility to perform the following functions:

- Import a repository. You can import a repository from the contents of the export file.

NOTE: When you are importing a custom repository (not the standard Siebel Repository), all languages which were part of the original repository are restored during import. For example, if you archive repositories weekly and your development repository contains support for both ENU and DEU, then both ENU and DEU are included when one of the archived repositories is imported.

- Export a repository. You can generate an export file in compressed format.
- Migrate a repository. For more information on using the configuration utility to migrate a repository, see [“Migrating Repositories and Schemas Between Databases” on page 942.](#)

If you need to back up the entire content of the Siebel database, use the database utilities provided by your RDBMS vendor.

NOTE: Whenever you make a change to the repository, compile all projects that belong to the latest version of the repository to create an updated .srf file. Keep a backup of the .srf file, so you can be sure .srf file truly reflects the contents of the updated repository.

When the Siebel Server and Database Server are installed, icons are created under the program group, which are called Configure DB Server and Configure Siebel Server. You only have to launch the Configure DB Server to invoke the command line you have written.

To import or export a repository under Windows

- 1** Launch the Database Server Configuration Utility by choosing Start > Programs > Siebel Enterprise Server *version_number* > Configure DB Server.

The Gateway Server Address screen appears.

- 2** Specify your Gateway Server Address and Enterprise Server Name and click Next.

The Installation and Configuration Parameters: Siebel Server Directory dialog box appears.

- 3** In the Siebel Server Directory dialog box, either accept the default value or choose the Browse button to select a directory, and then click Next.

The Installation and Configuration Parameters: Siebel Database Server Directory dialog box appears.

- 4** Either accept the default value or choose the Browse button to select a directory, and then click Next.

The Siebel Database Server Options: Siebel Database Operation dialog box appears.

- 5 In the Siebel Database Server Options: Siebel Database Operations dialog box, select Import/Export Repository from the list of operations, and then click Next.

The Import Repository Parameters: Select Repository Operation dialog box appears.

- 6 Choose Import Repository, and then click Next.

You can also choose the following options: Add language to an existing Repository, or Export Repository.

The Import Repository Parameters: Import Selection dialog box appears.

- 7 Choose Import Standard Siebel Repository or Import Custom Repository, and then click Next.

The following succession of dialog boxes appears:

Dialog Box	Description
Installation and Configuration Parameters: Language Selection	Select a language and click Next. English is the default language.
Installations and Configuration Parameters: RDBMS Platform	Select a RDBMS Platform. IBM DB2 UDB v7.1 is the default selection.
Installation and Configuration Parameters: ODBC Data Source Name	Select the ODNBC Data Source Name.
Installation and Configuration Parameters: Database User Name	Select the Database User Name and Database Password.
Installation and Configuration Parameters: Database Table Owner	Select the Database Table Owner and Database Table Owner Password.
Installation and Configuration Parameters: Import Repository Name	Select the name of the repository that you want to import and the filename from which you are importing.
Configuration Parameter Review	Summary dialog box gives a list of your choices. You can accept the configuration by clicking Finish.

To import or export a repository under UNIX

- 1 Source environment variables from `$SIEBEL_ROOT`.

`SIEBEL_ROOT` should be the path of your Siebel installation directory.

`LANGUAGE` should be set to the language in which the Configuration Wizard prompts appear; for example, `enu` for U.S. English.

If either of these values is incorrect or empty, reset them using one of the following commands, as appropriate to the shell you use:

```
setenv LANGUAGE New Value
```

or

```
export LANGUAGE|SIEBEL_ROOT=New Value
```

- 2 Depending on your shell, enter:

Korn shell

```
export SIEBEL_LOG_EVENTS trace3
```

C shell

```
setenv SIEBEL_LOG_EVENTS trace3
```

NOTE: Setting trace to 3, creates an appropriate level of detail in the log file for this activity.

- 3 Navigate to `$SIEBEL_ROOT /bin` and enter:

```
dbsrvr_config.ksh
```

This launches the Database Server Configuration Wizard.

- 4 Review the values of the following environment variables and confirm whether or not the settings are correct by entering either `Y` or `N`.

NOTE: If either the `SIEBEL_ROOT` or `LANGUAGE` value is not set or is incorrect, you must correct them before proceeding.

- 5** In the Siebel Server Directory dialog box, either accept the default value or choose the Browse button to select a directory, and then click Next.

The Installation and Configuration Parameters: Siebel Database Server Directory dialog box appears.

- 6** Either accept the default value or choose the Browse button to select a directory, and then click Next.

The Siebel Database Operation screen appears.

- 7** In the Siebel Database Operations screen, select Import/Export Repository.

The Import Repository Parameters: Select Repository Operation screen appears.

- 8** Choose Import Repository.

The Import Repository Parameters: Import Selection screen appears.

- 9** Choose Import Standard Siebel Repository or Import Custom Repository and click Next.

10 The following succession of screens appears:

Dialog Box	Description
Language Selection	Select a language and click Next. English is the default language.
RDBMS Platform	Select a RDBMS Platform. IBM DB2 UDB v7.1 is the default selection.
ODBC Data Source Name	Select the ODBC Data Source Name.
Database User Name	Select the Database User Name and Database Password.
Database Table Owner	Select the Database Table Owner and Database Table Owner Password.
Import Repository Name	Select name of the repository that you want to import and the filename from which you are importing.
Configuration Parameter Review	Summary dialog box gives a list of your choices. You can accept the configuration by entering Y.

NOTE: You follow the same procedure for Export Repository.

Using repimexp.exe for Importing, Exporting, and Creating a File Dump

The repimexp.exe program imports, exports, or creates a file dump of a repository. It can also do an INTL table import. INTL tables contain language-specific information and are a part of the repository.

You would rarely need to use repimexp.exe directly—you would use the configuration utility instead. The only circumstances when you would run repimexp.exe directly might be:

- You need to run repimexp.exe with special parameter settings that are inaccessible through the batch files.
- You need to perform a file dump.

To import a repository using `repimexp`

- In the command line, type the following:

```
repimexp /A I /G language_codes
```

where `language_codes` is a list such as ENU, FRA, JPN. Use ALL for all languages.

NOTE: If you want to import your repository with locale objects, you must specify at least one language code. Otherwise no locale objects will be imported. Therefore, when you compile the imported repository, it will not have any text in the user interface.

To export a repository using `repimexp`

- In the command line, type the following:

```
repimexp /A E argument_list
```

Export uses the arguments listed in [Table 60](#).

Table 60. Parameter Settings Passed as Export Arguments to `repimexp.exe`

Parameter	Required	Meaning
/U <userName >	Yes	Siebel administrator user name.
/P <password >	Yes	Siebel password.
/C <ODBC data source >	Yes	ODBC data source. The default is the value in the SIEBEL_DATA_SOURCE environment variable.
/D <table owner >	Yes	Siebel database table owner. The default is the value in the SIEBEL_TABLE_OWNER environment variable.
/W <language code >	Yes	Language mode, such as ENU or JPN. The default is SIEBEL_LANGUAGE. If this is not set, the default is ENU.
/R <repository >	Yes	Repository name. The default is Siebel Repository.

Table 60. Parameter Settings Passed as Export Arguments to repimexp.exe

Parameter	Required	Meaning
/1 < export repository userName >	Yes	Export repository user name. The default is the same as for /U.
/2 < export repository password >	Yes	Export repository password. The default is the same as for /P.
/3 < export repository ODBC data source >	Yes	Export repository ODBC data source. The default is the same as for /C.
/4 < export repository table >	Yes	Export repository table owner. Use siebel for Oracle and DB2, dbo for MS SQL Server. The default is the same as for /D.
/5 < export repository >	Yes	Export repository name. The default is Siebel Repository.
/B < appServer root >	Yes	Siebel Server installation directory to override SIEBEL_HOME environment variable.
/F < dataFile >	Yes	Data file, including path, to which to export.
/T < Y N >	No	Test only, do not export into database.
/V < Y N >	No	Verify data. The default is N.
/N < 0 1 2 >	No	Change creation and update information: <ul style="list-style-type: none"> ■ 0 = no change ■ 1 = change CREATED_BY, UPDATED_BY ■ 2 = change CREATED_BY, UPDATED_BY, dates columns The default is 1.
/M < Y N >	No	Commit changes even if verification fails. The default is N.
/L < logFile >	No	Log file for output messages.

To create a file dump using repimexp

- In the command line, type the following:

```
repimexp /A D /F <dataFile> [/L <logFile>]
```

To perform an INTL table import using `repimexp`

- In the command line, type the following:

```
repimexp /A X argument_list
```

INTL import uses the arguments listed in [Table 61](#).

Table 61. Parameter Settings Passed as INTL Import Arguments to `repimexp.exe`

Parameter	Required	Meaning
/G <language codes >	Yes	A list of language codes such as ENU, FRA, JPN. Use ALL for all languages.
/O <Y N>	Yes	Abort INTL import if unable to resolve parent row in server repository, that is orphans. The default is N.
/I <Y N>	Yes	Abort INTL import if insert fails. The default is Y.

Migrating Repositories and Schemas Between Databases

It is recommended that you have development and test environments that are isolated from the production environment.

The repository and user data need to be migrated in parallel between databases so the database schema for the user data, the business objects, and the user interface remain synchronized. Populate the test database, and when sufficient testing has taken place, migrate the repository and update the production database schema.

For information about setting up your system and database environment, see the *Siebel Server Installation Guide* for your operating system.

CAUTION: Do not migrate repositories between different versions of Siebel applications, as this will lead to an inconsistent environment.

If you configure only business object and user interface object definitions, you need to replace only the object definitions in the production repository with those in the development repository. Do this using the configuration utility, as described in [“Backing Up and Restoring Repositories” on page 934](#). Then distribute a new .srf file to client computers.

However, for upgrades involving schema changes, you need to use the configuration utility, which:

- Upgrades the data in the production server to the new schema
- Updates the repository object definitions

All mobile users need to synchronize prior to the upgrade and (if not using Siebel Anywhere) re-extract following the upgrade.

If you are using Siebel Anywhere, mobile users need to synchronize the next time they log on to their local database after the migration has occurred. Synchronizing will download new schema changes from the server to the mobile user’s local database. If they do not synchronize, there will be a mismatch between the local database and the server database.

It is recommended that you follow these major steps to migrate a repository and schema from one database to another. The first three steps are described in the following sections. The final step is an application administration task, and is described in *Applications Administration Guide*.

- Check in all projects—in both the source and target databases.

If you migrate a database schema with some projects still checked out, the migration will work but the project state will be not locked in the target database.

- Prepare the target database for the new repository.

The Target Repository parameter should be the name of a repository that does not already exist in the target database. Rename the current production repository if you already have one, for example, to Old Repository.

- Run the repository migration configuration utility.

NOTE: Siebel eBusiness Applications version 7.0 do not support customized database triggers. If you have created customized triggers on your Siebel base tables, you must disable them before migrating the schema. You will then need to recreate the triggers after the migration is finished.

- Upgrade mobile databases that are dependent on the target database.
- If you move a repository from one database to another, such as from development to test, you need to also re-create any new views, responsibilities, and list of values entries in the new environment.

Preparing the Target Database for the New Repository

Complete the following actions before you migrate the repository to the target database:

- Make sure that all mobile users perform a full synchronization to avoid any unexpected issues in a production environment as a result of database schema changes made to the new repository.
- Stop all server tasks and disconnect all database access until migration has been successfully executed.

NOTE: All connected users (including the database administrator) must disconnect before running the repository migration Configuration Utility.

- Do a full backup of the production database once all mobile user transactions have been merged.
- Make sure that the production database configuration meets the database requirements outlined in the Siebel Server installation guide for your operating system.

- Verify the names of all repositories in the target database.

You will later choose a new name that the repository being migrated will have in the target database. Siebel Systems recommends that you keep the name of your production repository constant. Accordingly, rename the existing production repository to show that it has been superseded. You will also later import a repository, to which you should give the standard name for your production repository.

Running the Repository Migration Configuration

NOTE: When you are migrating repositories over a wide area network (WAN) and running the Repository Migration Utility from the target environment, only the process of exporting the source repository to a flat file takes place on the WAN. All other processing takes place on the local area network (LAN) of the target environment.

The configuration utility does the following:

- Exports the designated repository from the source database
- Imports the designated repository into the target database
- Exports the logical schema definition from the specified repository to a .ddl file
- Synchronizes the physical schema of the target database with this logical schema definition
- (If you are using Siebel Anywhere) propagates new repository schema changes to mobile users

NOTE: If you have custom table spaces defined, the Database Server Configuration Utility used in the migration process is tablespace-aware.

To migrate a repository under Windows

- 1 Launch the Database Server Configuration Utility by choosing Start > Programs > Siebel Enterprise Server *version_number* > Configure DB Server.

The Gateway Server Address screen appears.

- 2 Specify your Gateway Server Address and Enterprise Server Name and click Next.

The Siebel Server Directory dialog box appears.

- 3 In the Siebel Server Directory dialog box, either accept the default value or choose the Browse button to select a directory, and then click Next.

The Siebel Database Server Directory dialog box appears.

- 4 In the Siebel Database Server Directory dialog box, either accept the default value or choose the Browse button to select a directory, and then click Next.

The RDBMS Platform dialog box appears.

- 5 In the RDBMS Platform dialog box, select the platform for your environment and then click Next.

The Siebel Database Operations dialog appears.

- 6 In the Siebel Database Operations dialog box, select Migrate Repository from the list of operations, and then click Next.

The following succession of dialog boxes appears.

- 7 Progress by completing the information in each dialog box, and then clicking Next.

Dialog Box	Description
ODBC Data Source Name	Enter the ODBC Data Source Name.
Database User Name	Enter the Database User Name and Database Password.
Database Table Owner	Enter the Target Database Table Owner and Table Owner Password.
Source Database Repository Name	Enter the Database Repository Name and the Target Database Repository Name.
Target RDBMS Platform	Select the Target RDBMS Platform. IBM DB2 UDB v7.1 is the default.

Dialog Box	Description
Target Database Encoding	Select whether or not the target database is Unicode.
Target Database ODBC Datasource	Enter the Target Database ODBC Datasource.
Target Database User Name	Enter the Target Database User Name and Password.
Target Database Table Owner	Enter the Target Database Table Owner and Table Owner Password.
Index Table Space Name (DB2-specific)	If you choose IBM DB2 for the Target Database Platform, you get this dialog box. Enter the Index Table Space Name and 4 KB Table Space Name.
16 KB Table Space Name (DB2-specific)	If you choose IBM DB2 for the Target Database Platform, you get this dialog box. Enter the 16 KB Table Space Name and 32 KB Table Space Name.
Index Table Space Name (Oracle-specific)	If you choose Oracle as the Target DB Platform, you have only two questions about Index and Table space. There are no 4 KB, 16 KB, and 32 KB tablespaces in Oracle. For Microsoft SQL server, there are no dialog boxes about tablespaces.
Configuration Parameter Review	Summary dialog box gives a list of your choices. You can accept this configuration by clicking Finish.

To migrate the schema under UNIX

- 1** Source environment variables from \$SIEBEL_ROOT by typing:

```
source siebenv.csh
```
- 2** Set the following environment variables:
 - SIEBEL_ROOT should be the path of your Siebel eBusiness Application installation directory.

- LANGUAGE should be set to the language in which the Configuration Wizard prompts appear; for example, enu for U.S. English.

If either of these values is incorrect or empty, reset them using one of the following commands:

- `setenv LANGUAGE ENU` (where ENU represents your display language)
- `setenv SIEBEL_ENTERPRISE <Enterprise Name >`

- 3** Navigate to `$SIEBEL_ROOT /bin` and enter:

```
dbsrvr_config.ksh
```

This launches the Database Server Configuration Wizard.

- 4** Review the values of the following environment variables and confirm whether or not the settings are correct by entering either Y or N.

- SIEBEL_ROOT
- LANGUAGE

NOTE: If either the SIEBEL_ROOT or LANGUAGE value is not set or is incorrect, you must correct them before proceeding.

- 5** Specify the path of your Siebel Server root directory, or accept the default by pressing ENTER.
- 6** Specify the path of your database server root directory, or accept the default by pressing ENTER.
- 7** Enter the number that corresponds to your database platform.
- 8** From the Select Repository Operations menu, choose Migrate Repository (4).

9 Progress by completing the information in each screen.

Dialog Box	Description
RDBMS Platform	Select an RDBMS Platform. IBM DB2 UDB v7.1 is the default.
Target Database Encoding	Select whether or not the target database is Unicode.
ODBC Data Source Name	Enter the ODBC Data Source Name.
Database User Name	Enter the Database User Name and Database Password.
Database Table Owner	Enter the Target Database Table Owner and Table Owner Password.
Source Database Repository Name	Enter the Source Database Repository Name.
Target Database Repository Name	Enter the Target Database Repository Name.
Target RDBMS Platform	Select the Target RDBMS Platform. IBM DB2 UDB v7.1 is the default.
Target Database ODBC Datasource	Enter the Target Database ODBC Datasource.
Target Database User Name	Enter the Target Database User Name and Password.
Target Database Table Owner	Enter the Target Database Table Owner and Table Owner Password.
Index Table Space Name (DB2-specific)	If you choose IBM DB2 for the Target Database Platform, you get this dialog box. Enter the Index Table Space Name and 4-KB Table Space Name.
16K Table Space Name (DB2-specific)	If you choose IBM DB2 for the Target Database Platform, you get this dialog box. Enter the 16-KB Table Space Name and 32-KB Table Space Name.

Index Table Space Name (Oracle-specific)	If you choose Oracle as the Target DB Platform, you have only two questions about Index and Table space. There are no 4K, 16K, and 32K tablespaces in Oracle. For Microsoft SQL server, there are no dialog boxes about tablespaces.
Configuration Parameter Review	Summary dialog box gives a list of your choices. You can accept this configuration by entering Y.

NOTE: Updating statistics with a full table scan is the job of the DBA once any repository migration, upgrade, or installation finishes.

Upgrading Mobile Databases

Follow these steps:

1 Restart Siebel Remote processes.

If you have mobile users in your target database environment, restart the Application Server Processes, regardless of whether you are using Siebel Anywhere.

When you have restarted the processes, wait until the Transaction Pre-Processor and the Transaction Router have processed all pending transactions before proceeding with the remaining steps.

2 Regenerate local database templates.

Use the Siebel Server component Generate New Database to regenerate the local database template file to update its schema to the same version as the database server.

3 Re-extract mobile users.

If you are not using Siebel Anywhere to upgrade your mobile clients, re-extract all mobile users, using the Database Extract Component.

Make sure that you have copied the new database template to all production Siebel Remote Servers before re-extracting the mobile users. The `distmpl.ksh` (check) helps you copy the database template.

NOTE: However, if mobile databases are not reextracted, users will still be able to synchronize—no error message will be generated. This is to allow Siebel Anywhere, which users might use to upgrade mobile databases, to continue working.

If you are using Siebel Anywhere, refer to *Developing and Deploying Siebel eBusiness Applications* for instructions on propagating schema extensions.

Creating Patches

A *patch file*, like an archive file, consists of exported object definitions. The difference between a patch file (.spf file) and an archive file (.sif file) is that the patch file contains two versions of each object definition, one from the pre-upgrade source repository and one from the post-upgrade. (An archive file contains only one version of each object definition, and all object definitions are from the same repository.)

Figure 241 shows how pre- and post-upgrade versions of an object definition are paired in the patch file, and then utilized in applying the patch to the target repository.

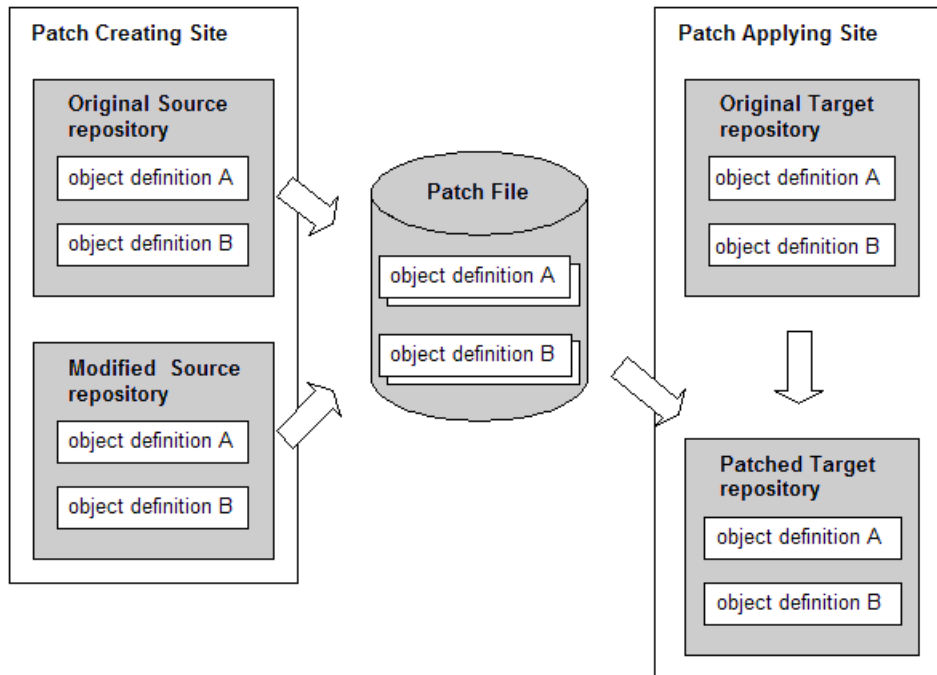


Figure 241. How a Patch Works

The pair of pre- and post-release object definitions in the patch file provide *before* and *after* snapshots of the object definition. The patch application process considers both when determining what changes to make to the target repository.

Creating a Patch File

A wizard steps you through the process of creating a patch.

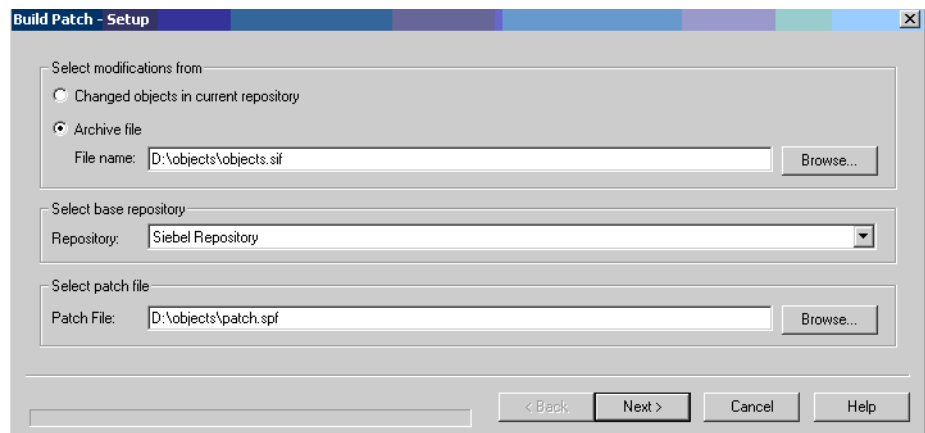
To create a patch file

- 1 Make sure that both the original source and the modified source repositories are present on the client computer.
- 2 Select File > Open Repository from the menu bar and open the modified source repository.

NOTE: You can skip this step if you are building a patch file from an archive file.

- 3 Select Tools > Utilities > Build Patch from the menu bar.

The Build Patch - Setup window appears.



- 4 In the “Select modifications from” radio button group, select either Changed objects in current repository or Archive file:

- **Changed objects in current repository.** Allows you to generate the set of source object definitions in the patch file from all object definitions in the currently open (modified source) repository that have a value of TRUE in their Changed property. The Changed property indicates changes to property values or child object definitions for all object definitions that have changed since a specified date. This is an easy way to capture all object definitions that have changed since the start of work on the new release.

NOTE: This is useful for creating cumulative patch files—that is, if several patches are created over time, each successive patch will include all changes that went into previous patches plus the most recent changes, as long as the Changed Indicator Date has not been modified. This is the real reason that it is possible to define a patch using the Changed property. If you use the Changed indicator in this or any other way, you need to be careful that the Changed Indicator Date does not get set arbitrarily.

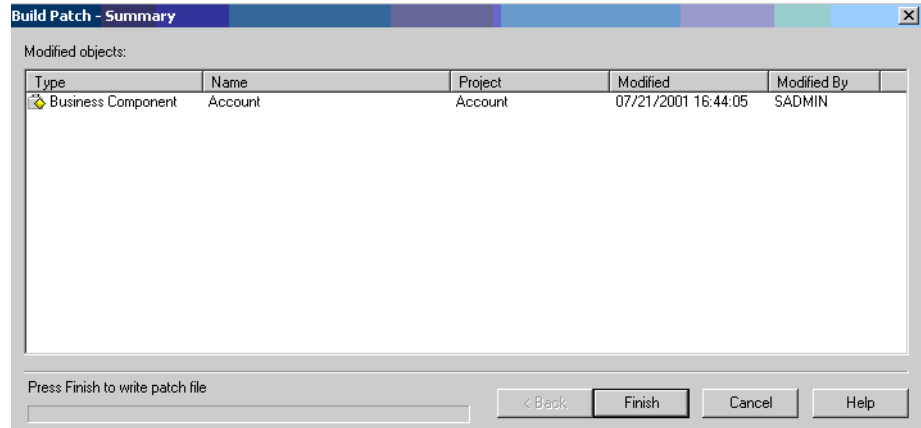
- **Archive file.** Allows you to use an existing archive file to generate the same set of object definitions in the patch file. Use this option when the set of patch object definitions is identical to a recently exported archive file, or when you want to explicitly select individual top-level object definitions to be included. In this latter case, generate the archive file prior to generating the patch file. Building a patch from an archive file may also be preferable when there are too many object definitions with a Changed value of TRUE.

If you selected the Archive File option, specify a pathname and filename for the archive file in the File Name box, or click the Browse button and navigate to the archive file.

- 5 In the Repository box, select the name of the original source repository.
- 6 In the Patch File box, specify a pathname and filename for the patch file to create.

7 Click Next.

The Build Patch Summary window appears.



If you selected the Archive file option, the list of object definitions for the patch loads immediately.

If you selected the Changed objects option, the system requires a minute or more to generate the list, because it needs to scan through the repository and check all the Changed property values.

8 Click Finish.

The patch file is generated in the directory location you specified in [Step 4 on page 922](#).

Applying a Patch File

The patch upgrades the repository to which it is applied similarly to how the Application Upgrader does. The difference is that you do not have the opportunity to override the default conflict resolution rules. A conflict only occurs if an object property changes in both the source and the target repositories simultaneously.

For example, if you create a new Account field based on an extension column in the target repository, and then apply a patch from the source repository that includes the Account business component, the new field will not be overwritten in the target repository because the same new field has not been added in the source.

If you change the sort specification of the Account business component in the target repository, and the sort specification has not changed in the source, the new sort specification in the target will remain. However, if the sort specification has changed in both the source and the target, then a conflict arises for which a resolution is required.

The default conflict resolution rules can be read from the repository by looking at the Type object in Siebel Tools. It has a child object type called Attribute, which has a property called Siebel Wins (or Standard Wins in the Object List Editor). If this is set to TRUE, the value in the source repository is accepted. If FALSE, the value in the target repository is accepted.

To apply a patch

- 1** In Siebel Tools, choose Tools > Utilities > Apply Patch.

The Select Patch to Apply dialog box appears.

- 2** Select the Siebel Patch (.spf) file, and then click Open.

The Apply Patch - Preview window appears, and the patch is opened.

- 3** Click Next.

The Apply Patch - Summary window appears. The patch is loaded, the patch objects are compared to their corresponding repository objects, and then the patch is applied.

- 4** Click Finish to exit.

Integrating with External Source Code Control Software

You can optionally interface your repository check in/check out mechanism in Siebel Tools with a third-party source code-control system such as Microsoft Visual SourceSafe. When source control integration is enabled, each time a project is checked into the server repository, an archive file containing all the object definitions in the project is also checked into the source control system. As a result, successive versions of the project are maintained in the source control system.

If you want to revert to an earlier version of a project, you can check out the project archive file from the source control system to your local machine. Then you check out the project from the server, import the archive file into the local repository (overwriting the object definitions locally), and check the project back into the server repository.

Enabling the Interface

You enable and partly configure the interface to an external source control system using the Development Tools Options dialog box.

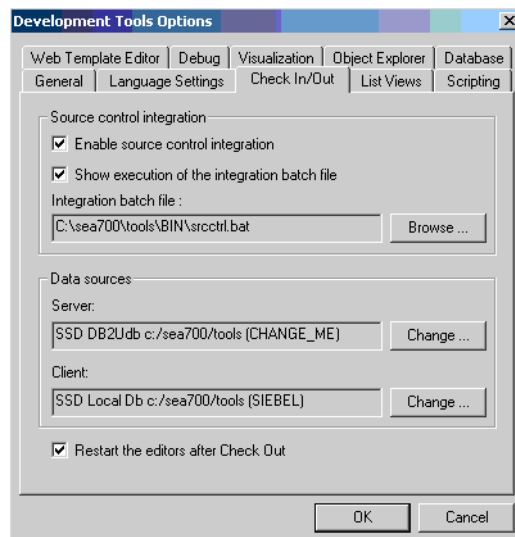
To enable the source control interface

- 1 Select the Tools > Options menu option.

The Development Tools Options dialog box appears.

- 2 Select the Check In/Out tab.

The Check In/Out tab in the dialog box appears.



- 3 Click the Enable Source Control Integration check box to set it to TRUE.
- 4 Click the “Show execution of the integration batch file” check box to enable this feature. A DOS window is launched in the foreground when the srcctrl.bat batch file is executed. This feature is for diagnostic purposes and facilitates debugging a customized batch file.

- 5 If the `srcctrl.bat` file is in a different directory location than the one indicated, type in a different location or select it using the Browse button.

By setting the Enable Source Control Integration check box to TRUE and specifying a `srcctrl.bat` batch file, you are informing the system that it is to generate an archive file for each project when performing repository check-in, and to run the batch file at the conclusion of repository check-in. The batch file executes command-line interface commands that are specific to the source code control software you are using. Each command-line command launches the external source control software with instructions to check an archive file or files into the source control system.

Archive files for source code control have the same format and `.sif` filename extension as an archive file generated with the Export feature. One `.sif` file is generated for each project checked in to the server repository.

Configuring the `srcctrl.bat` File

The `srcctrl.bat` batch file contains the sequence of commands to be executed in order to check the archived projects in to the source control system. The name of the archive file for the project to be checked in is specified as an argument to the batch file, in addition to other arguments. The syntax for the command line that executes the batch file is as follows:

```
SRCCTRL action dir comment_file project_file
```

The arguments for `srcctrl.bat` are shown in [Table 62](#).

Table 62. Arguments for the `srcctrl.bat` File

Argument	Description
<i>action</i>	Check in or check out.
<i>dir</i>	Pathname of the directory on your local file system where the items are located.
<i>comment_file</i>	Contains the comment text to be provided to the source control software with the project file.
<i>project_file</i>	Name of the archive file for one project, enclosed in double quotes.

Srcctrl.bat executes once for each project, following the completion of repository check-in. It checks the archive file for the project into or out of the source control system. Srcctrl.bat is executed from a command line that is internally generated from the Siebel application software. You do not have access to the command line setup, and you cannot modify the parameter list.

The following batch file program code is taken from the standard srcctrl.bat file provided with Siebel applications, and is designed to work with Microsoft Visual SourceSafe. Comment lines have been removed. You need to customize the program code in this batch file, particularly if you are running source control software other than Microsoft Visual SourceSafe, or if the path is incorrect.

```
set PATH=C:\Program Files\DevStudio\Vss\win32\;%PATH%

set SOFTWARE=ss

set CHECKIN=%SOFTWARE% checkin

set CHECKOUT=%SOFTWARE% checkout

set ADD=%SOFTWARE% add

set SETPROJ=%SOFTWARE% cp

set PROJECT=$/PROJPOOL

set SRC_USR=

set SRC_PSWD=

set OPTIONS=-i-y -y%SRC_USR%,%SRC_PSWD%

set COMMENT=-c@

set NON_COMMENT=-c-

set FILE=

set LOGFILE=C:\Temp\xml.log

echo
=====srcctrl.bat===== >>
%LOGFILE%

set ACTION=%1

shift
```

```
set DIR=%1

shift

set COMMENT=%COMMENT%%1

shift

set FILE=%1

echo Change local directory to %DIR% >> %LOGFILE%

chdir %DIR% >> %LOGFILE% 2>&1

echo Set %PROJECT% as the working folder at Source Control System
>> %LOGFILE%

%SETPROJ% %PROJECT% >> %LOGFILE% 2>&1

if errorlevel 100 goto END

if %ACTION%==checkout goto CHECK_OUT

if %ACTION%==checkin goto CHECK_IN

:CHECK_OUT

echo =====Check out file %FILE% from Source Control
System=====

if not exist %FILE% echo "New File" >> %FILE%

attrib +r %FILE%

echo Add %FILE% in case it doesn't exist in Source Control System
>> %LOGFILE%

%ADD% %FILE% %NON_COMMENT% %OPTIONS% >> %LOGFILE% 2>&1

echo Start checking out %FILE% from Source Control System >>
%LOGFILE%

%CHECKOUT% %FILE% %NON_COMMENT% %OPTIONS% >> %LOGFILE% 2>&1

goto END

:CHECK_IN
```

```
echo =====Check in file %FILE% into Source Control
System=====

echo Check in %FILE% into Source Control System >> %LOGFILE%

%CHECKIN% %FILE% %COMMENT% %OPTIONS% >> %LOGFILE% 2>&1

attrib -r %FILE%

goto END

:END

echo =====End Of srcctrl.bat=====
>> %LOGFILE%
```

[Table 63](#) explains the variables used in the srcctrl.bat batch file.

Table 63. Variables in srcctrl.bat

Variable	Description
PATH	Identifies the directory where the source code control software is installed. Modify this setting to reflect its actual location on your machine.
SOFTWARE	Source control system's command line utility. The command line utility for Microsoft Visual SourceSafe is "ss".
CHECKIN	Command at the start of the command line that calls for check-in into the source control system.
CHECKOUT	Command at the start of the command line that calls for check-out from the source control system.
ADD	Command at the start of the command line that calls for adding files in the source control system.
SETPROJ	Command at the start of the command line that calls for setting the working folder in the source control system.
PROJECT	Project (working folder) in the source control system where the items will be checked in/checked out.
COMMENT	Command-line Comments clause for each of the files being checked in or out. This is generated from the Comment argument to the batch file.
OPTIONS	Text of the Options clause to include in a command line.

Table 63. Variables in srcctrl.bat

Variable	Description
SRC_USR	User logon name to include in the Options clause. This is a source control software user name, not the user name for a Siebel application.
SRC_PSWD	User password to include in the Options clause. This is a source control software password.
FILE	Filename of the archive file, obtained from the argument list of the batch file. This file needs to be checked in or out.
LOGFILE	Path and filename of the log file that will be generated.

You need to:

- Change the program code in the batch file to reflect the development environment
- Distribute to all of the developers at your site

These MS-DOS limitations exist in Windows 95 and 98:

- The default command-line character limitation is 127 characters. You can increase the global command-line character limit to its maximum by placing the following line in config.sys:

```
shell=c:\windows\command.com /u:255
```

- The default environment space limitation is 256 bytes. You can increase it by placing the following line in config.sys:

```
shell=c:\command.com /p /e:2048
```

or by placing the following line in the [NonWindowsApp] section in the system.ini file:

```
CommandEnvSize=2048
```

2048 is the size of the new environment space, so you might specify a different value here.

Microsoft Visual SourceSafe Examples

The following sections provide you with examples for using Microsoft Visual SourceSafe.

Check In Example

You have two projects checked out that you want to simultaneously check in to the server and to the source control software. The projects selected are “Project A” and “Project B.” The latest version of Project A.sif in Visual SourceSafe is 6, and the latest version of Project B.sif is 5.

When you click the Check In button, the following sequence occurs:

- 1 Project A and Project B are checked in to the server repository.
- 2 C:\sea7xx\tools\bin\srcctrl.bat is invoked. This carries out steps 3, 4, and 5.
- 3 Project A.sif and Project B.sif are checked out and locked in Visual SourceSafe.
- 4 Project A is exported to C:\sea7xx\tools\temp\projects\Project A.sif, and Project B is exported to C:\sea7xx\tools\projects\Project B.sif.
- 5 Project A.sif and Project B.sif are checked in to Visual SourceSafe. The version numbers are incremented so that the latest version of Project A.sif in Visual SourceSafe is version 7, while Project B.sif is version 6.

Revert to Previous Version Example

Consider the situation in which an erroneous definition of Project A has been checked in to the server repository. This is stored in Microsoft Visual Source Safe as version 5 of Project A.sif. You want to revert to version 4 of Project A, since that does not contain the errors.

- 1 Check out version 4 of Project A.sif from Visual SourceSafe into C:\sea7xx\tools\temp.
- 2 Check out Project A from the server repository.
- 3 Import Project A.sif into the local repository using the Overwrite option to resolve object definition conflicts. This replaces the existing definition of Project A with the one in the archive file.

- 4 Check Project A in to the server repository. Project A.sif is automatically checked in to Visual SourceSafe as version 6.

Check In/Check Out Options (Source Control Integration)

The Check In/Out tab in the Development Tools Options dialog box (Figure 242) provides options for setting up check-in and check-out.

The Source control integration options are discussed below; for information about the Data sources options, see “[Check-In/Check-Out Options \(Data Sources\)](#)” on [page 1005](#).

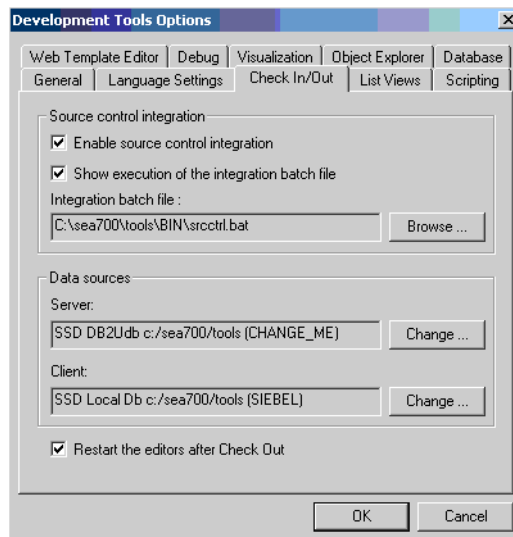


Figure 242. Check In/Out tab in Development Tools Options window

The Source control integration options are the following:

- **Enable Source Control Integration check box.** Set this check box to TRUE and specify the location of the srcctrl.bat batch file in the Integration batch file text box if you want to generate an archive file for each project when performing repository check in, and at the conclusion of repository check in to run the batch file once for each project.
- **Show execution of the integration batch file check box.** Set this to TRUE to launch a DOS window in the foreground when the srcctrl.bat batch file is executed. This feature is for diagnosis purposes and facilitates debugging a customized batch file.
- **Integration batch file text box and browse button.** Specifies the location of the srcctrl.bat batch file used by Siebel applications to instruct the source control software to provide check in or check out of archive files.

Upgrading Repositories: Siebel Application Upgrader

The Siebel Application Upgrader reduces the time and cost of version upgrades by allowing you to acquire new features from the latest release while preserving the custom configuration changes made to the current repository. It notifies system administrators about conflicts between object customizations and new releases, automatically merges differences between object definitions, and allows you to manually override and apply any changes.

The Siebel Application Upgrader allows you to upgrade custom configurations to new releases by merging them with a current Siebel eBusiness software release. This capability minimizes the cost of application upgrades and allows you to quickly deploy production versions of Siebel eBusiness Applications. For more information, see the upgrade guide for your operating system.

The Application Upgrader allows you to accomplish the following:

- Determine what has changed with new releases of Siebel eBusiness Applications
- Compare custom configurations with new changes delivered in a new Siebel release

- Choose which changes to apply, whether made by your company’s developers or by Siebel Systems in the new release

NOTE: The Application Upgrader is for merging an entire customized repository with a standard one. To merge portions of repositories, use the Import/Export or Patch features.

Figure 243 shows the Application Upgrade-Objects List.

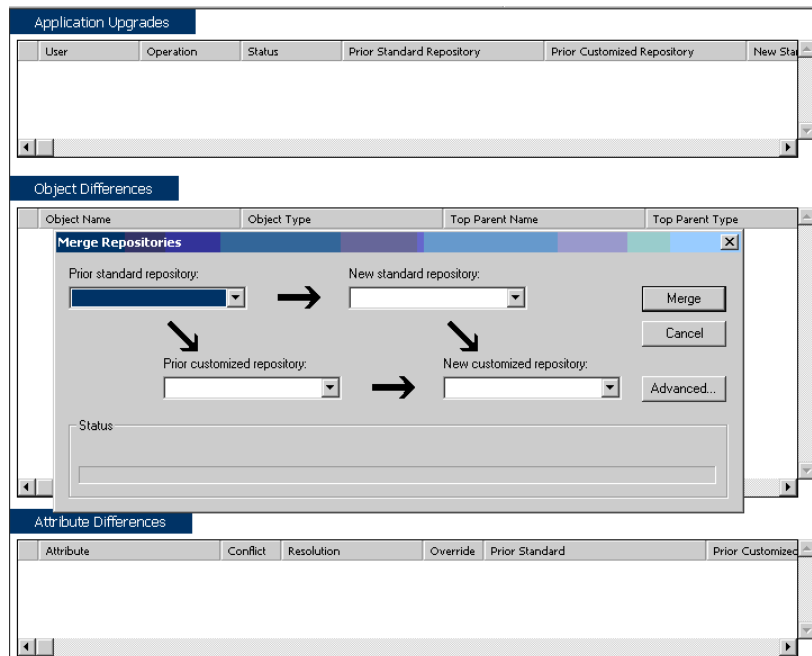


Figure 243. Siebel Application Upgrader

For more information about the Application Upgrader, see the upgrade guide for your operating system.

Web Client Migration Wizard

The Web Client Migration Wizard allows you to convert UI elements from Win32 clients connected to a server or local database to the Web client. You can move your configurations and users to the Web Client interface when you upgrade to Siebel Web applications. To support this migration, a new feature is included for converting applets and views into Web applets and view applets. In particular, this applies to new applets and views that you have created and may have customized. The conversion process is comprised of two parts:

- 1** The bulk conversion of All New and Customized Applets and Views that is performed immediately after the Repository Merge. This is initiated from the Application Upgrader screen.
- 2** The next step is to review the migration and to redo it selectively for certain applets or views. This is initiated from the Applets and Views list applet in the Object List Editor.

You are able to select and enable Web customizations of multiple applets or views using Web Client Migration Wizard, which helps convert applets and views that you have created.

For more information, see the upgrade guide for your operating system.

Automatic Upgrade of Copied Objects

Siebel Tools allows copied objects to inherit some of the behavior of their ancestors, which makes it easier to upgrade Siebel applications, reduces the time and cost of adjusting an application after an upgrade, and also supports parallel development by allowing some frequently used objects to be copied.

Certain repository objects that are copied during configuration can be upgraded with a new property called Upgrade Ancestor that stores the name of the ancestor object. This allows copied objects to be upgraded in the same way as the ancestor objects from which they were copied. Thus when you copy an existing object, you can specify its upgrade ancestor; during an upgrade the copied object will be upgraded the same way as the original. This feature is available only for objects of type Applet, Business Component, Report, and Integration Object.

Upgrade Inheritance functionality:

- The Upgrade Ancestor property stores the name of the ancestor object (that is, the one from which the current object was copied).
- If the Upgrade Ancestor property is not null, you can upgrade the copied object as if it were the ancestor object.
- No special action is taken during import even if the “Upgrade Ancestor” property is specified, for this is specific to application upgrades. But imported objects can have this property set. When the next application upgrade is done, the property is taken into account.
- Inheritance does not apply to patch files. The “Upgrade inheritance” property is applicable only during application upgrades. Its not taken into account during the application of a patch and no action is performed based on this property.
- During the merge, the newly created objects are given all the changes corresponding to their ancestor. Objects with the Upgrade Ancestor property include:
 - Applets
 - Business Components
 - Integration Objects
 - Reports

Basically, you can create a copy of an existing object (applets, business components, integration objects, and reports) and specify an Upgrade Ancestor.

Upgrade Inheritance Scenario

For example, you may want to make a copy of the Account List applet and call it the Premium Account List applet. This new applet may differ from the original one in that it has a special search specification that is displayed only in those accounts that are considered premium accounts. In a subsequent release, Siebel Systems may add new out-of-the-box list columns to the Account List applet. During an application upgrade, your Account List applet and the Premium Account List applet will retain the configuration changes you made. However, both applets will receive the new out-of-the-box list columns added in the new version because of Upgrade Inheritance functionality. Without this new feature, the copied applet would not receive the new list columns during the upgrade process.

Recommended Guidelines for Copying Objects

The guidelines for copying objects vary. If the object is a UI object such as an applet or view, it should only be copied if significant changes will be made to the look and feel of the object. Copying the object rather than using the out-of-the-box object makes certain that the modified look and feel would be preserved following the upgrade. If only minor changes will be made to the UI object, it is better to use the out-of-the-box object since this will eliminate the time spent on configuration and the continuing maintenance of the repository. Other valid reasons for copying UI objects would be:

- If two different UI objects had to display different records (that is, different search specifications on applets).
- If different read/write properties between two objects were necessary (that is, one applet is read-only, and the other is editable) and if this could not be accomplished through the dynamic read-only buscomp user property.
- If different drilldowns were necessary for different applets depending on the view that contains them (and provided this could not be accomplished through a dynamic drilldown).

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

- The copied applet *must* be used with the original business component, not a copy of the original business component.

- If you want to use a copied applet with a copied business component, you need to change the class of the copied applet.

These guidelines are illustrated in the following example:

Applet	Applet Class	Business Component	Business Component Class
Quote List (SCW)	CSSFrameListQuote	Quote	CSSBCQuote

If you copy the Quote List (SCW) applet, the copied applet can only use the original Quote business component, not a copy of the Quote business component. However, if you change the class of the copied Quote List (SCW) applet from CSSFrameListQuote to CSSFrameList, then you can use the copied Quote List (SCW) applet with a copy of the Quote business component (which is still based on the CSSBCQuote class).

If the object is a non-UI object, such as a business component or business object, it is recommended that you not copy them if not absolutely necessary. These recommendations pertain mainly to the copying of BCs, which is more common than copying BOs. Copying BCs and BOs should be avoided. In other words, the out-of-the-box object should be used, and modifications should be made to this object rather than creating a copy and using the copy in the configuration. There are some situations when the copying of BCs cannot be avoided. The following reasons are given for this:

- When a BC must appear twice in a business object; for example, when the Account BC and Sub Account BC both appear within the Account BO.
- When two BCs contain different search specifications *and* predefault values for the type field that differentiates the records of these two BCs.

These guidelines are aimed at minimizing the use of copied BCs because of the problems that can occur with repository maintenance, specialized classes, and upgrading. Even with the Upgrade Inheritance feature it is important to be careful in deciding to use copied objects.

How Enhancements Are Applied During an Upgrade

During upgrades, it is very common that objects in the repository are changed. For example, an applet might have a few list columns added or a business component might have some fields and a multi-value link added. To do this, the objects that need to be changed during the upgrade are recognized by their Name property. For example, you would query the repository for the Account BC and add the necessary new items to it. If you did not have the Upgrade Inheritance feature and the Account BC had been copied as Acme Account, you would not recognize the new BC as a copy of the Account BC and would not add the required changes to the copy during the upgrade. These additions might be minor, but often these omissions can cause numerous application errors after the upgrade and can be time-consuming to detect and correct.

During an upgrade, the Upgrade Inheritance feature makes sure that copied objects receive the same changes that are applied to the object from which they were copied. This is done automatically by the upgrader, and there is no manual step involved except for specifying the property.

NOTE: This functionality is applied only to the following object types: *business component, applet, integration object, and report*.

Repository Location of the Upgrade Ancestor

During the application upgrade the contents of three repositories are compared to produce the final, post-upgrade repository which contains both the customizations made by the customer as well as any enhancements that were added by engineering during the upgrade. The three repositories compared are the following:

- Prior Standard Repository: Ancestor Repository.
- Prior Customer Repository: Ancestor Repository that has been customized by the client.
- New Standard Repository: New Siebel Repository.

The Upgrade Ancestor object of a copied object must exist in the New Standard repository in order for any enhancements to be applied to descendants during the merge. The repository produced is the following:

- New Customer Repository: New Siebel Repository customized by the client.

Ancestor objects must be found in the repository in order to be applied to their descendants during a merge.

Configuration Steps for Upgrade Inheritance

UI objects should be copied if the look-and-feel of the application will change significantly or if there is a difference needed between two objects (that is, different search specifications on two applets). Business components should only be copied after all other configuration approaches have been exhausted, and copying is clearly the only solution. The issues involving repository maintenance and specialized classes still are present with copied business components. Upgrade Inheritance functionality allows certain copied objects to be upgraded and inherit the same characteristics that the parent object possessed. This avoids post-upgrade errors and configuration problems.

After creating the copied object, specify the parent object name in the Upgrade Ancestor property of the copied object. This is what allows the copied object to be recognized as a copy during the application upgrade, and it is changed along with its parent object.

NOTE: You must manually populate this property because it is not automatically populated when you copy an object. Remember that this property can only be populated if the copied object is an applet, business component, integration object, or report object type.

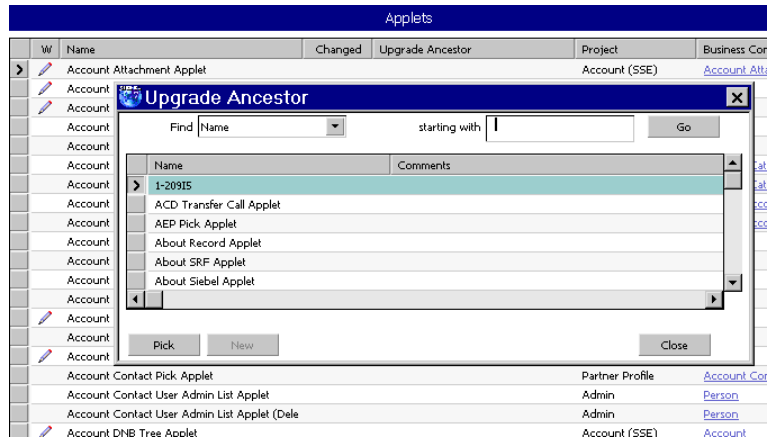
To copy an applet, business component, integration object, or report object and children

- 1 Select the Object type (Applet, Business Component, Integration Object, or Report) in the Object Explorer.
- 2 Select an entry in the Object list applet.
- 3 Choose Edit > Copy Record to create a copy of this record.
- 4 In the new record, fill in a new name in the Name field.

- 5 Click the ellipsis in the Upgrade Ancestor field.

The Upgrade Ancestor pick list appears.

- 6 Select a value, and then click Pick.



The picklist shows all the other business components, applets, integration objects, and reports in the repository.

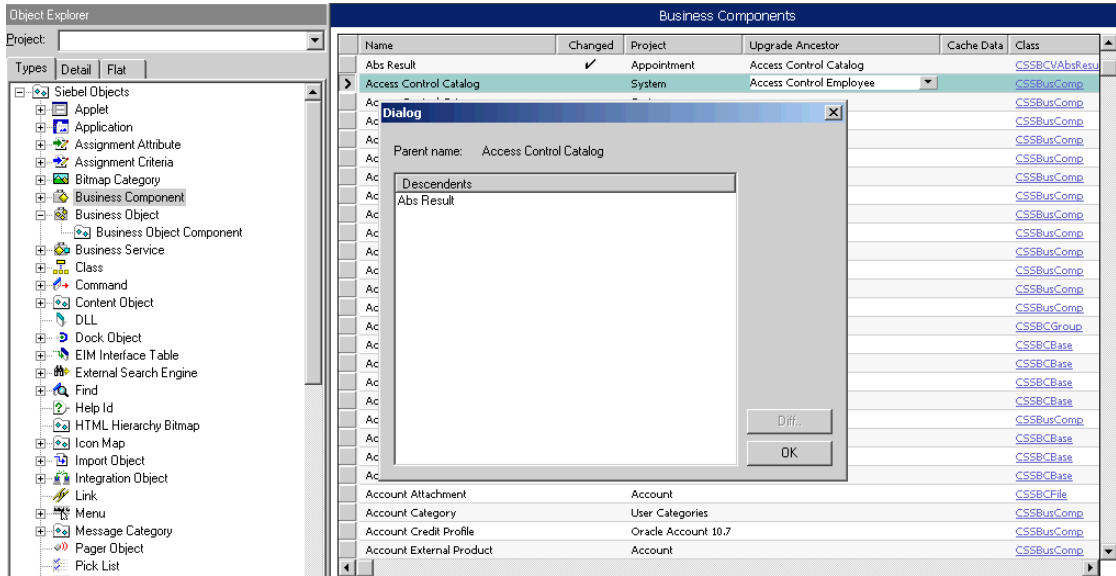
To view all descendants or copies of an object

- 1 Right-click an object.

A dialog box is displayed.

2 Select View Descendants from the dialog box.

A dialog box appears showing the Parent Name and Descendants.



Propagating Changes from a Parent to Descendants

The new Object Comparison and Synchronization feature allows two objects to be compared, and the differences between the two objects applied to one another to keep them synchronized. See the section, “[Object Comparison and Synchronization](#)” on page 975.

Object Comparison and Synchronization

You can view a side-by-side comparison of any two objects of the same type. Differences are visually highlighted through color-coded icons. You can select and copy properties and individual child objects from one object to the other.

Using this feature, you can propagate a change made to an ancestor object to its descendants or other objects of a similar types. Differences between similar objects can easily be assessed and adjusted. You can also compare properties of checked-out objects with their counterparts on the server.

Viewing the Object Comparison Dialog Box

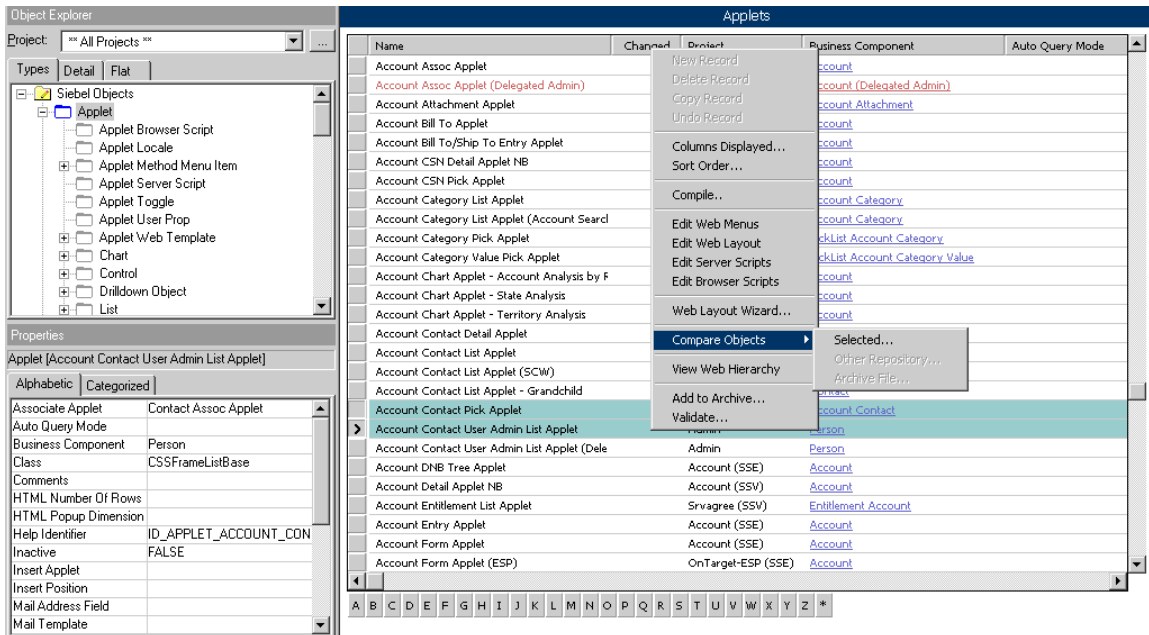
The Object Comparison dialog box displays the differences between the selected objects. First, you should select the two objects you would like to compare.

To view differences between two objects

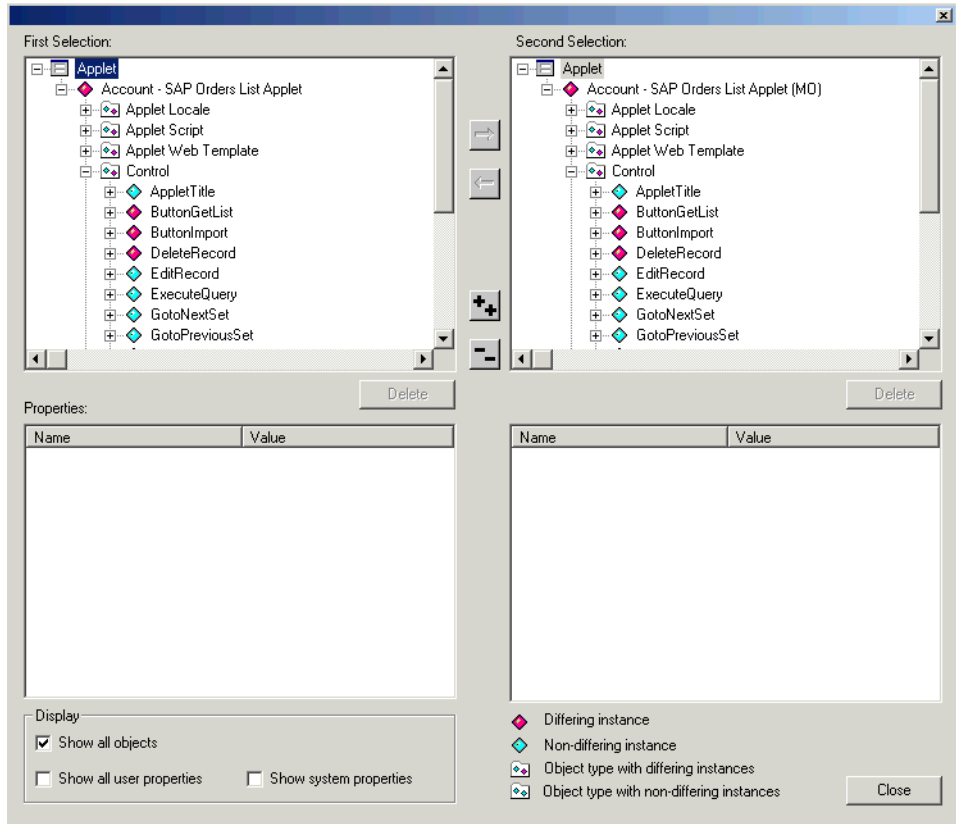
- 1 Select any two top-level objects of the same object type.

Select the second one using SHIFT.

- 2 Choose Tools > Compare Objects > Selected. (Selected-Repository, Selected-Archive, and Archive-Archive are the other options.) Compare Objects is also available by right-clicking as shown in the following example.



The Object Comparison dialog box appears. This dialog box shows the differences between two objects.



The following features are important in the Object Comparison dialog box:

Explorer Applets. The Explorer applets in the upper left and right sections of the dialog box are similar to what you see after clicking the Detail table of the Object Explorer. They are always in sync in order to show a line-by-line comparison.

Child objects that do not exist in either tree applet are represented with placeholders.

Expanding and collapsing folders are synchronized between the two tree applets. If you expand or collapse an object in one tree applet, its counterpart is automatically expanded or collapsed in the other applet.

Icons. Distinctive icons are used visually to identify child objects that either have no counterpart or have conflicting properties with their counterparts.

- Instances where the two objects do not have property differences are marked by a blue diamond icon. In the example AppletTitle and List for both objects have blue diamonds and are therefore the same.

- Instances where there are differences are marked by a red diamond icon. In the example, ButtonGetList and ButtonImport have differences. The highlighted control for the objects being compared is ButtonGetList. ButtonGetList for the Account - SAP Orders List applet has a Method Invoked value of SAPGetList. The value for the Account - SAP Orders List (MO) applet is SAPGetList (MO).

Properties. By default, only those properties that are different are shown. You need to check the Show All User Properties check box to view all the properties (Figure 244). The Show All Objects check box refers to objects on the Tree controls. By default all objects are shown (those with and without any property differences), and Show All Objects is checked. Show System Properties check box refers to some specific properties like Created, Created By, Updated, Updated By, and so on.

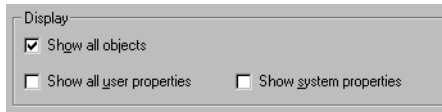


Figure 244. Properties Dialog Box

Copying and Deleting. The dialog box has features that allow you to copy and delete specific instances while comparing objects. For example, you can choose a specific Control from the ACD Transfer Call Applet and click Delete. Copying of objects to reconcile the differences between two objects is also accomplished by using the arrows.

NOTE: These operations can only be performed on objects that belong to locked projects.

Arrows. You can copy from one object to the other using the left and right arrows.

- The buttons marked with two plus or two minus signs are used to expand or contract the whole tree. The button marked with two minus signs contracts the whole tree.

- When you select an object instance, the right pointing arrow is enabled if you have the object on the right locked. The left pointing arrow is enabled if you have the object on the left locked.
- When you click the right-pointing arrow, the selected objects in the left tree applet are copied to the object in the right tree applet. If the objects do not exist in the right tree applet they will be created. If they do exist, the objects in the right tree applet will have their properties changed to reflect those in the left tree applet. When you click the left-pointing arrow, the same pattern will occur.
- When you copy an object from one tree applet to the other, the children of the object are copied as well.

NOTE: These arrows are disabled if the destination object does not have its project locked.

Deleting an Object.

- You can delete an object by selecting it and clicking the Delete button under the selected tree applet.
- You can delete only one object at a time.
- The Delete button is only active when you lock the object and when you have selected a single object instance. If you want to delete specific instances of an object, a dialog box is displayed showing the changes and asking you, “Are you sure you want to perform the operation?” If you click No, no changes will be made. If you click Yes, the deletion is made.
- Read-only objects and child objects are greyed out and cannot be updated.

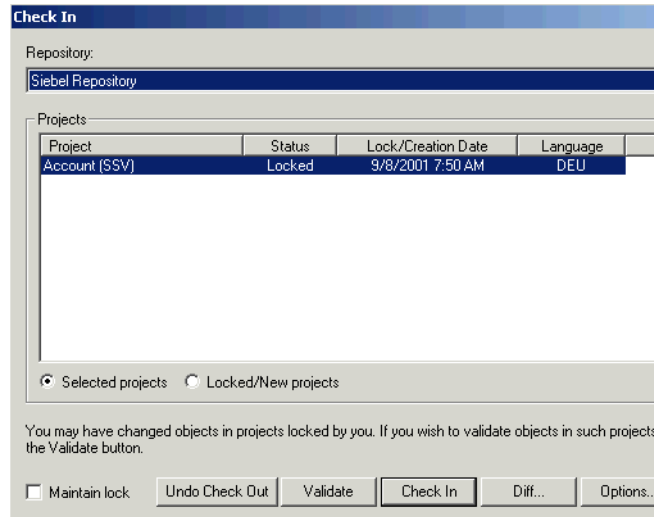
Differences Between Checked-Out Projects

The Object Comparison dialog box can also be used to view the differences between projects that are checked out and their counterparts on the server. Multiple projects can be selected and their differences can be viewed at the same time.

To view differences between checked-out projects and those on the server

- 1 Choose Tools > Check In.

The Check In dialog box appears.



- 2 From the Check In dialog box, select the two projects you want to compare and click the Diff... button.

The Object Comparison dialog box appears with the selected projects.

You are not allowed to perform Copy or Delete operations in this situation.

NOTE: Undo Check Out button allows you to undo all the checkouts.

Entering the Comparison

You can make comparisons between any two objects of the same type in the following locations.

NOTE: These options are available in the Compare Objects submenu that is available when you click the right mouse button after selecting one or more objects. They are also available under Tools > Compare Objects. The Compare Objects dialog box can also be launched from the View Descendants dialog box. You can compare the ancestor and one descendant or the ancestor and many descendants.

- 1 Two objects that are in the same repository (see [“Option One”](#) below)
- 2 One object is in the current repository and the other is in another repository in the same database (see [“Option Two”](#) on page 983)
- 3 One object is in the current repository and the other is in an archive file (see [“Option Three”](#) on page 984)
- 4 Both objects are in archive files (see [“Option Four”](#) on page 985)

Option One

If you select any two objects in the Object List Editor and right click to access a menu item called Compare Objects > Selected, the only option that is available is Selected, which is described below:

Selected Option: Allows you to make comparisons between two selected objects from the same repository. After you right-click the two objects and choose Selected, the Compare Objects dialog box is launched.

Option Two

If you select only one object and choose Tools > Compare Objects, one of the options you get is the following:

Selected vs. Archive Option: A Select Archive File to Compare Against dialog box opens (Figure 245), and you can select a .sif file that will be used for comparison with the object selected in the Object List Editor (OLE).

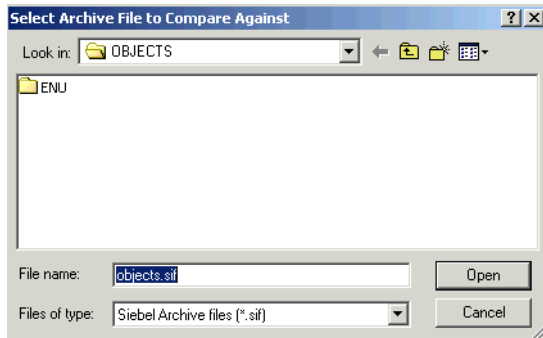


Figure 245. Select Archive File to Compare Against Dialog Box

If a corresponding object type is not found in the archive file, the user will receive a message indicating this and the Object Comparison dialog box will not open. Otherwise, the comparison will be conducted starting at the project level.

Option Three

If you select only one object and choose Tools > Compare Objects, one of the options you get is the following:

Selected vs. Repository Option: You will be presented with a dialog box with a list of repositories in current database (Figure 246). You select one and click OK or Cancel.

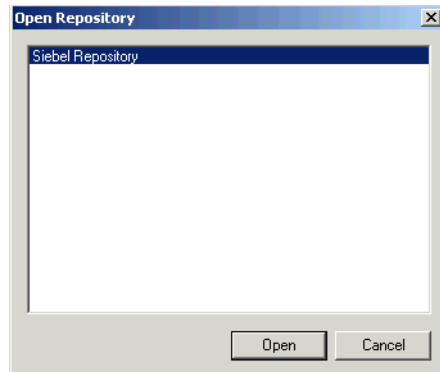


Figure 246. Open Repository Dialog Box

When you pick a repository and click OK, Tools finds a corresponding object with the same name. The Object Comparison dialog box opens with the object in the current working repository displayed in the left applet and the corresponding object in the selected repository in the right applet.

You will be able to update the current working repository or the selected repository from the Object Comparison dialog box if you have the appropriate projects locked in both repositories.

Option Four

If you select only one object and choose Tools > Compare Objects, one of the options you get is the following:

Archive vs. Archive Option: You are able to compare two archive files at the project level by selecting this option.

When you select this menu item, you are presented with a Select Left File for Left Side of Comparison dialog box that allows you to select an archive file (Figure 247). When you select an archive file and click OK, the Object Comparison dialog box opens with the left side populated with the contents of the selected archive file.

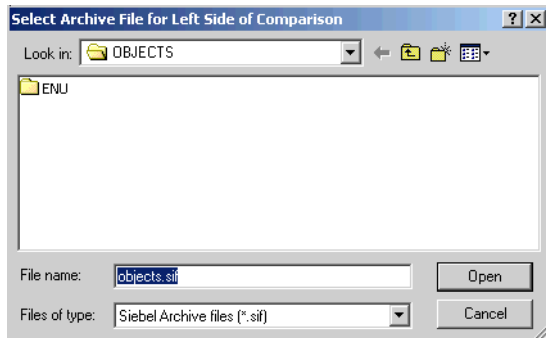


Figure 247. Select Archive File for Left Side of Comparison Dialog Box

The Select Archive File for Right Side of Comparison dialog box is then displayed so you can select another file and click OK. The right side of the Object Comparison dialog box is populated with the contents of the second file.

When comparing the two files, both will be read-only.

Application Development Projects **18**

This chapter defines Siebel projects and explains how to:

- Get information about projects
- Create projects
- Assign object definitions to projects
- Check out and check in projects
- Compile projects

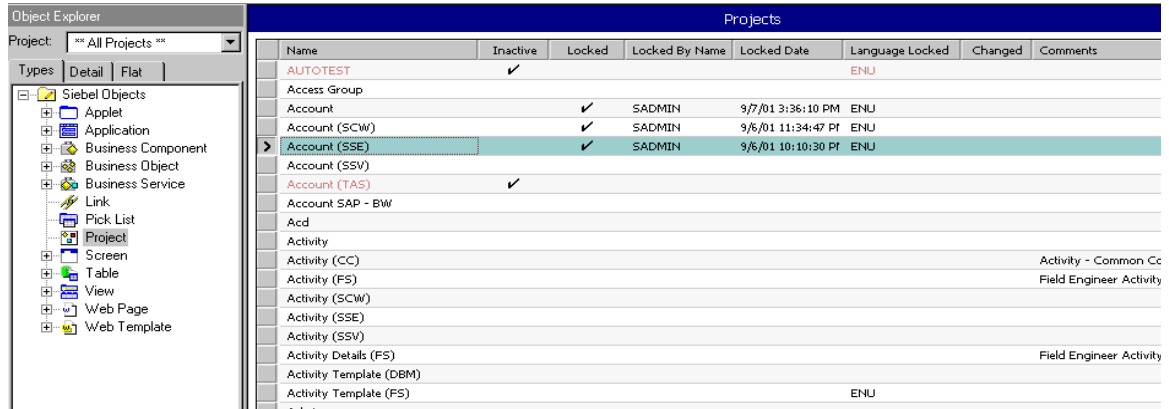
What Are Siebel Projects?

Projects are named sets of object definitions that reside in the Siebel repository—mechanisms to meaningfully group object definitions so they can be worked on by teams of application developers.

The master copy of the repository resides on a server database. Multiple developers can access it to make changes and additions by locking and checking out groups of server repository object definitions to local (client) repository databases for modification, and checking them back in to the server following modification and testing.

A Siebel application is delivered with a large number of existing projects.

Figure 248 shows a list of projects displayed in the Object List Editor window.



Name	Inactive	Locked	Locked By Name	Locked Date	Language Locked	Changed	Comments
AUTOTEST	✓				ENU		
Access Group							
Account		✓	SADMIN	9/7/01 3:36:10 PM	ENU		
Account (SCW)		✓	SADMIN	9/6/01 11:34:47 PM	ENU		
Account (SSE)		✓	SADMIN	9/6/01 10:10:30 PM	ENU		
Account (SSV)							
Account (TAS)	✓						
Account SAP - BW							
Acid							
Activity							
Activity (CC)							Activity - Common Cc
Activity (FS)							Field Engineer Activity
Activity (SCW)							
Activity (SSE)							
Activity (SSV)							
Activity Details (FS)							Field Engineer Activity
Activity Template (DBM)							
Activity Template (FS)					ENU		

Figure 248. List of Projects Displayed in the Object List Editor Window

The names of projects that are delivered with a standard Siebel application indicate the functional area with which they are associated. For example, Account contains definitions pertaining to the Account functional area. A name without a suffix (like Account) usually contains Business Object Layer definitions that span multiple Siebel applications.

Project names that have a suffix (for example, Account(SSE)) contain user interface or business object definitions that are specific to the Siebel application indicated by the suffix. The suffix SSE in Account(SSE) (fourth entry from the top in [Figure 248 on page 988](#)) indicates an entry containing Account user interface data for the Siebel Sales application.

Other suffixes indicating user interface data only are SSV for Siebel Service and CC for Siebel Call Center.

Other suffixes indicating both user interface and business object data are FS for Field Service, TAS for Targeted Account Sales, UI for User Interface, and DBM for Database Management.

Getting Information About Repositories and Projects

This section discusses:

- Selecting the current repository
- Getting information about the current repository
- Viewing object definitions by project

Selecting the Current Repository

Under normal circumstances there is only one repository available on your local database, and one available on the server database for your development workgroup. Typically this repository (in either location) is called the Siebel repository and is opened by default when you open Siebel Tools and log on to Local or Server. However, there are circumstances—especially when your group is in the process of upgrading to a new version of Siebel eBusiness Applications—in which multiple repositories can be present, especially on the server.

Whenever there is a possibility of multiple repositories, choose File > Open Repository and verify that you already have the correct repository open, or select a different one.

The File > Open Repository option lists only the repositories that have been previously checked out from the server. In order to see an additional server repository, you must first do a Get to get all of the projects for that repository in the Check Out window. This is described in [“Getting Information About the Current Repository” on page 990](#).

Getting Information About the Current Repository

You can get version, compilation, and path information about the current repository by choosing Help > About SRF from the menu bar in Siebel Tools. The About Repository File window is shown in [Figure 249](#).

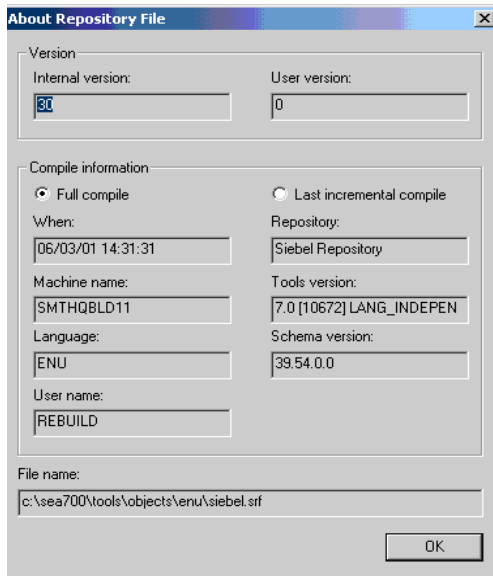


Figure 249. About Repository File Window

This window displays the following information:

- **Internal version.** Version number maintained internally at Siebel that changes only when the internal format of the .srf file changes, such as at the time of a major release. It has no significance for customer developers.
- **User version.** Reserved for use by Siebel Anywhere, which maintains this number when kits are created that upgrade the .srf file. The value is read when a version check occurs.

- **Full compile/Last incremental compile radio buttons.** Determines whether the Compile Information fields display information about the most recent full compilation or incremental compilation. If there have been no incremental compilations since the full compilation, the latter option is dimmed (as shown in [Figure 249 on page 990](#)).
- **When.** Date of the last compilation—incremental or full, as specified in the radio buttons.
- **Machine name.** Name of the client computer on which the .srf file was compiled.
- **Language.** Language code of the language specified for user interface translation.
- **User name.** User name (that is, the Microsoft Windows logon name) of the user who compiled the repository.
- **Repository.** Repository name of the repository that was current when the compilation was run, generally Siebel repository.
- **Tools version.** The version number and build number of the Siebel Tools software used to compile the repository. This is useful information for Siebel Technical Services if they are helping you in resolving a problem with your configuration.
- **Schema version.** Database schema version of the database from which the repository was compiled.
- **File name.** Name and path of the .srf file being used internally to define the Siebel Tools application, generally located in C:\sea7xx\tools\objects.

NOTE: This is not the same Siebel application .srf file that is produced by compilation and distributed to client machines. Siebel Tools itself is a customized Siebel application, so it has its own repository file.

Viewing Object Definitions by Project

To restrict the objects displayed in the BusObject Designer to those that belong to a particular project, select an entry from the project picklist you access at the top of the Siebel Tools Object Explorer window (shown in [Figure 250](#)).

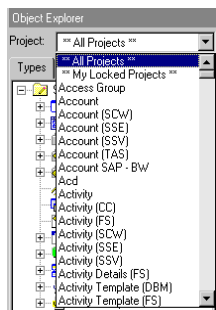


Figure 250. Project Picklist

To see all projects, select the All Projects entry from the top of the picklist.

The Object Explorer displays object types only if there are definitions of that type in the selected project. [Figure 251 on page 993](#) shows the applets in the Account (SSE) project.

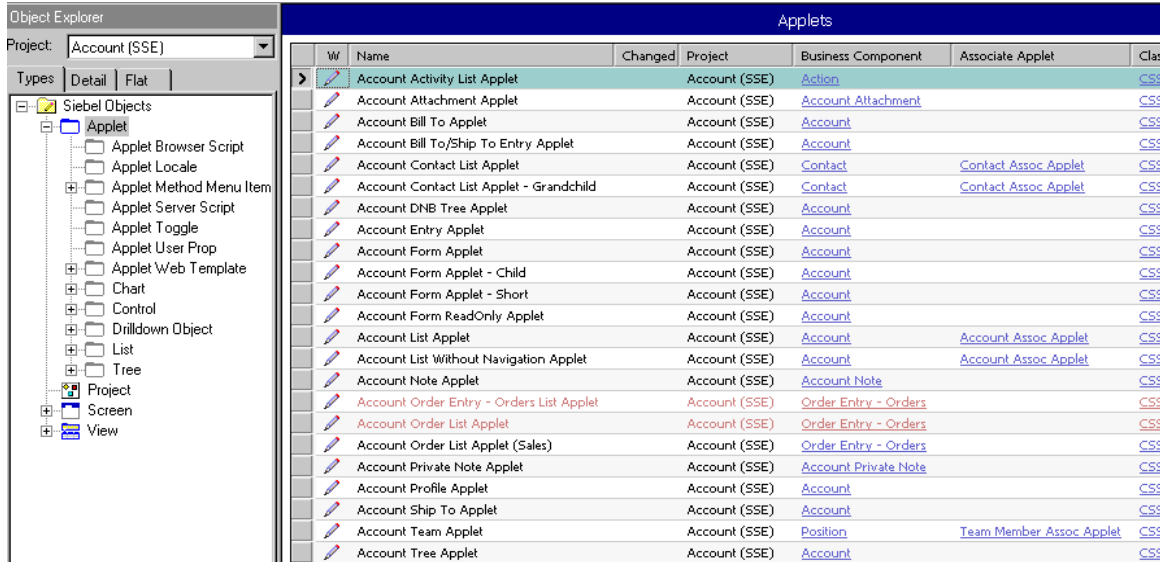


Figure 251. Applets in the Account (SSE) Project

Getting Projects

The process of copying all projects from the repository on the server database to your local development database is known as a full get. Doing a full get differs from checking out projects (see [“Checking Out Projects”](#)) in the following ways:

- A full get does not lock the projects on the server database.
- A full get copies all projects from the server repository to your local repository. You cannot select individual projects.
- A full get will override all projects on your local database, whether they are locked or not locked.

You must perform a full get to populate a newly initialized local database with a copy of all projects from the server repository. You cannot compile before doing a full get because your .srf files must be based on the comprehensive set of Siebel object definitions plus any changes you make. You can also use a full get to refresh your local repository with the repository stored on the server.

NOTE: The sample database, unlike the local database, cannot receive checked-out object definitions, and its object definitions cannot be checked in to the server database. The sample database is strictly for instructional use.

By default the full get process performs database commits in regular intervals during the get process rather than a single commit at the end of the get process. You can disable this option by choosing View > Options, selecting the Check In/Out tab, and then clearing the *Enable incremental commit during Full Get* check box.

To do the initial get of all projects from the repository

- 1** Choose Tools > Check Out...

The Check Out dialog box appears (see [Figure 252 on page 996](#)).

- 2** Choose the name of your development repository from the Repository picklist.

NOTE: The repository that you select is not necessarily the one opened by Siebel Tools.

- 3** Select the All Projects radio button.
- 4** Click Options.
- 5** In the Development Tools Options window, make sure your Server Data Source is pointing to your server development database and your Client Data Source is pointing to the local database you previously initialized and are currently running against.
- 6** Close the Development Tools Options window.

- 7 In the Check Out dialog box (see [Figure 252 on page 996](#)), click Get.

After the get is complete, your currently open local repository has the same contents as the server repository from which you did the get.

Checking Out Projects

Project Check In/ Out is a source control mechanism that allows you to coordinate development work with multiple developers. When you check out projects from the server repository, the project is locked on the server and is copied to your local repository. This allows you to make changes to the project and prevents others from working in the same project.

Checking out a project differs from a full get in the following ways:

- You can select the projects you want to check out.
- Projects you check out are locked on the server.

NOTE: Check Out must be performed against the server database from which the local database has been extracted.

To check out a copy of one or more projects

- 1 Select Repository > Check Out.

The Check Out dialog box appears (see [Figure 252 on page 996](#)).

- 2 Make sure that the correct repository is selected.
- 3 Select the projects you want to modify.
- 4 Click the Options button.
- 5 In the Development Tools Options window (see [Figure 255 on page 1006](#)), make sure the Server and Client data sources are specified correctly.
- 6 Close the Development Tools Options window.

- 7 In the Check Out dialog box, click Check Out.

The contents of the object definitions you have checked out are not automatically displayed or redisplayed in the Object List Editor following check out. Select View > Refresh Windows to display updated information.

CAUTION: Password encryption interferes with check out. If you will be checking out projects, you need to disable password encryption in the client or CFG file when running Siebel Tools.

Check Out Dialog Box

The Check Out dialog box is shown in [Figure 252 on page 996](#).

You can check out projects only in the language Siebel Tools runs in. Check the current Siebel Tools language mode by choosing View > Options and clicking the Language Settings tab.

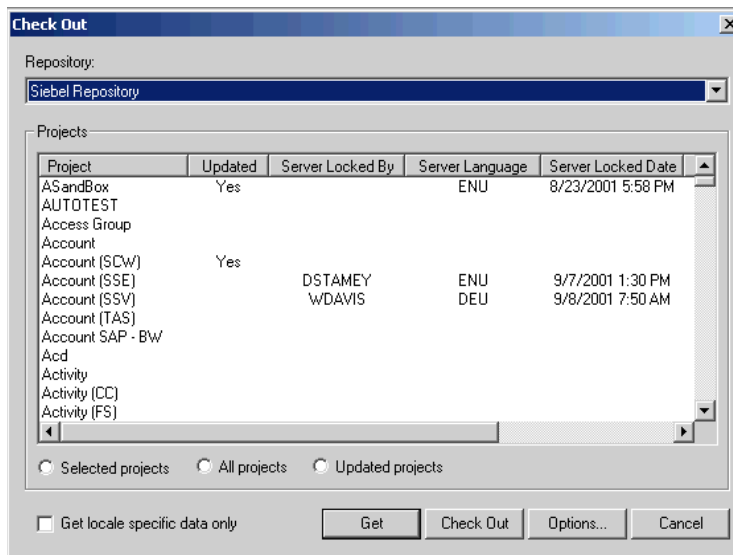


Figure 252. Check Out Dialog Box

User interface elements in the Check Out dialog box are:

- **Repository picklist.** Available repositories on the server. The list of projects in the projects list table reflects the list of projects in the selected server repository. If you select a different server repository from the one currently open in Siebel Tools locally, a warning appears, and you must either get all projects or change the repository selection.

- **Projects list.** Projects in the server repository.

Click a project or projects to check out or get.

The projects list contains the following list columns:

- **Project list column.** Displays the name of each project in the server repository.
- **Get Locale specific data only.** Gets only the locale (language) specific data in selected projects, in the language specified during Siebel Tools installation.
- **Server Language.** The language of the project currently locked on the server. Only one language can be locked at one time.
- **Client Language.** The language of the project currently locked on the client. Only one language can be locked at one time.
- **Updated.** A value of Yes appears if the server Locked By and Locked Date are different from the client version, indicating that your version of the project is out of sync with the server's version.
- **Server Locked By.** Logon ID of the developer who currently has this project checked out on the server.
- **Server Locked Date.** Check-out date.
- **Client Locked By.** Logon ID of the developer who currently has this project locked locally.
- **Client Locked Date.** Date the project was locked locally.
- **Selected projects radio button.** When this radio button is checked, you can select individual projects to check out or get.
- **All projects radio button.** When this radio button is checked, all projects in the repository are selected to check out or get.

- **Updated projects radio button.** When this radio button is active, only projects with an Updated value of Yes are selected. This allows you to check out or get only those projects on the server that are new or different from corresponding projects in the local repository. Normally you will get rather than check out such projects, to bring your local repository up to date.
- **Get button.** Selected projects are copied to the local repository, replacing pre-existing versions there. The lock status of these projects is not changed on the server. You can get any projects on the server, including those locked by others.
- **Check Out button.** Copies all object definitions in the selected projects to the local repository and locks them on the server (and client). You cannot check out projects that are currently locked on the server (if you select a locked project in the projects list, the Check Out button dims).
- **Options button.** Opens the Development Tools Options dialog box with the Check In/Out tab selected. This is the same dialog box that appears when you choose Tools > Options from the menu bar.
- **Cancel button.** Cancels check-out and closes the Check Out dialog box.
- **Get Local Specific Data Only check box.** Checking this box gets string translations and locale-specific attributes being stored in the locale objects.

Creating New Projects

You can create a new project by selecting the Project object type in the Object Explorer (Types tab) and creating a new record in the Object List Editor.

In general, you would add new projects:

- To break large numbers of object definitions into more manageable groups.
- When developers are contending for different sets of object definitions in the same projects.

CAUTION: Once you create a project you cannot delete it from Siebel Tools, but you can do it through SQL.

Renaming Projects

You need to rename projects on the server, not on the local database.

CAUTION: Never change the name of a project that standard Siebel Object definitions are associated with.

To rename a project or reassign object definitions on the server

- 1** Make sure developers have checked in all checked-out projects.
- 2** Open the server repository.
- 3** Change the Name property of the project or the Project properties of the relevant object definitions.
- 4** Have developers do a get of all projects on the server repository.
- 5** Have developers do a full compilation the next time they compile.

Assigning Object Definitions to Projects

Every object definition must be assigned to one and only one project, either:

- A Siebel-supplied project, or
- A user-created project

You assign an object definition to a project by setting the Project property.

The Competitor (SME) business component shown in [Figure 253](#) (the selected business component) has an assigned Project property of Oppty (SME).

Business Components					
W	Name	Changed	Project	Cache Data	Class
	Abs Result		Appointment		CSSBCVAbsResult
	Accept/Reject		Acid		CSSBCVAcceptReject
	Access Control Catalog		System		CSSBusComp
	Access Control Category		System		CSSBusComp
	Access Control Child Group		System		CSSBusComp
	Access Control Employee		System		CSSBusComp
	Access Control Employee Manager Po		System		CSSBusComp
	Access Control Group		System		CSSBusComp
	Access Control Group Member Person		System		CSSBusComp
	Access Control Manager Position		System		CSSBusComp
	Access Control Org Member Person		System		CSSBusComp
	Access Control Parent Org		System		CSSBusComp
	Access Control Party Reporting Relatic		System		CSSBusComp
	Access Control Test		System		CSSBusComp
	Access Group		Access Group		CSSBCGroup
	Access Group Member		Access Group		CSSBCBase
	Account		Account		CSSBCBase
	Account (Delegated Admin)		Admin		CSSBCUser
	Account - Get SAP Order List Header		SAP Account		CSSBCBase
	Account - Get SAP Order List Item		SAP Account		CSSBCBase
	Account - Get SAP Order List Output		SAP Account		CSSBCBase

Figure 253. Project Property of a Business Component

Moving Object Definitions Between Projects

To move an object definition between two projects

- 1 Check out both projects to your local database.
- 2 Change the Project property of the object definition you intend to move, so that it reflects the name of the new project.
- 3 Check in both projects together.

CAUTION: Failure to check in both projects at the same time can leave the repository in an inconsistent state.

- 4 Inform other developers that they must do a simultaneous get of the two projects prior to doing any subsequent work on the object definition in either project.

NOTE: If you delete an object from a particular project, and then re-create it with the same name in another project, this has the same effect as moving the object between projects. Therefore, you should take the same precautions.

In general, you should not delete object definitions. Deleting objects might cause parts of your application not to work. If you do not want to use an object, you should set the value of its Inactive property to TRUE.

Checking In Projects

Use Check In to copy new or modified object definitions in a local repository to the server repository.

The check-in process:

- Replaces any existing versions of the checked-out object definitions with modified versions, and adds any new ones
- Unlocks the project

If you use Check In with Maintain lock, the server repository is updated, but the project is still locked to your local repository.

When checking in projects, consider the following:

- Password encryption interferes with check-in. You need to disable password encryption when running Siebel Tools if you will be checking in projects.
- You can only check in projects under the same working language in which you checked them out.
- Check In must be performed against the server database from which the local database has been extracted.

To check in new or modified object definitions

- 1** Choose Repository > Check In...

The Check In dialog box appears (see [Figure 254 on page 1003](#)).

- 2** Click the Options button.

In the Development Tools Options dialog box, make sure the server and client Data Sources are pointing to the correct databases.

- 3** Close the dialog box.

- 4** In the Check In dialog box, select the appropriate repository in the Repository picklist.

- 5** Do one of the following:

- Click the Selected projects radio button and select the individual projects to check in.
- To check in all locked and new projects, click the Locked/New Projects radio button.

- 6** Click Check In.

CAUTION: Depending on the size of the project, the check-in process might require some time. Do not interrupt the process, as doing so can leave your repository in an unstable state. If for any reason the check-in process is interrupted, you must redo it. This completes the unfinished tasks and unlocks the project on the server.

Check In Dialog Box

Figure 254 shows the Check In dialog box. You can check in any previously checked-out project in any language.

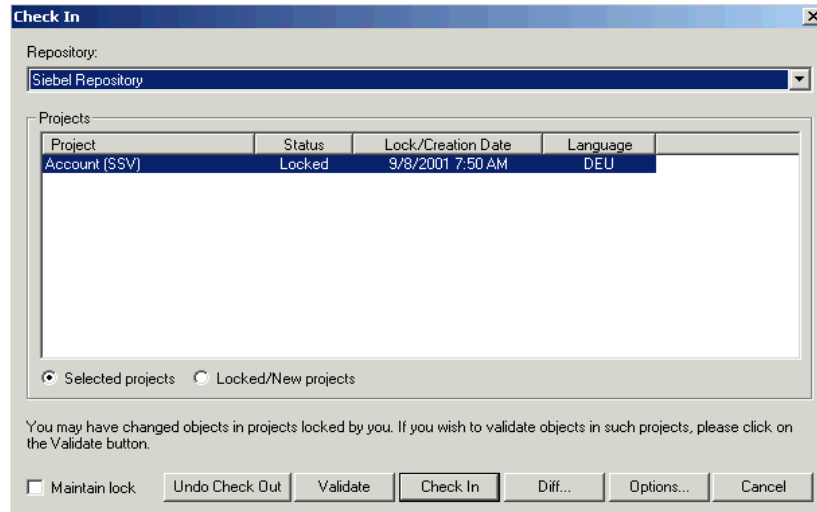


Figure 254. Check In Dialog Box

User interface elements in the Check In dialog box:

- **Repository picklist.** Available repositories in the local database. The list of projects in the Projects list table reflects the list of projects in the selected repository (in addition to locally created projects).
- **Projects list table.** Identifies all projects in the local repository selected in the Repository picklist. It contains the following list columns:
 - **Undo check out.** Does not check in object definitions to the server. This releases the lock on the server, so that another developer can work on those objects.
 - **Language.** The language of the project checked out.

- **Project.** Displays the name of each new or checked-out project in the local repository. Projects obtained by means of get operations are not listed, because these are not available for check-in. (You can check in only projects that you have previously checked out or created locally.)
- **Status.** Contains the value New or Locked for each project, indicating whether you created it yourself or obtained it through check-out.
- **Lock/Creation Date.** Displays the date and time when you created the project or checked it out from the server.
- **Language.** Displays the language in which the project was checked out.
- **Selected projects radio button.** When this radio button is checked, you can manually select individual projects to check in.
- **Locked/New projects radio button.** Selects all of the projects in the list—that is, all projects you have created or obtained through check-out.
- **Maintain lock check box.** When checked, tells the system to check in the projects, but keep them locked on the server.
- **Check In button.** Initiates the check-in process.
- **Diff button.** Opens the Project Differences window for comparison of the object definitions in the projects you are checking in with their corresponding versions on the server. For more information, see [“Determining Project Differences at Check-In Time” on page 1008](#).
- **Options button.** Opens the Developer Tools Options window where you specify check-in/check-out settings, especially server and client data source names.
- **Cancel button.** Closes the Check In dialog box.
- **Validate button.** Validates selected projects.

Check-In Guidelines

Here are some rules to follow when you check in projects:

- Before doing a check-in, make sure that the projects you are checking in are in a stable state and have been thoroughly tested against your local repository. Check in projects only when all dependent Siebel VB code is complete.

- Check in all dependent projects at the same time to be sure that the configuration on the server remains consistent.

For example, if you create a new PickList object definition in the PickList project and reference that object definition in your Oppty project, check in both projects to the server at the same time.

- Consider the timing of your check-in and its impact on 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 because another developer's configurations may depend on a particular feature you have added to your project.

Check-In/Check-Out Options (Data Sources)

The Check-In/Out tab in the Development Tools Options dialog box ([Figure 255](#)) provides options for setting up check-in and check-out.

The Data sources options are discussed below; for information about the Source control integration options, see [“Check In/Check Out Options \(Source Control Integration\)”](#) on page 965.

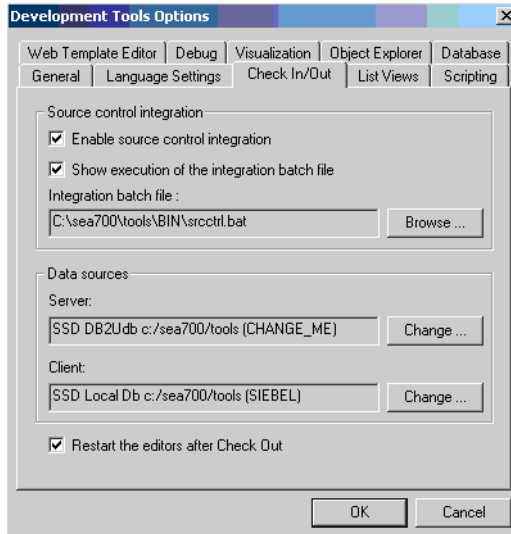


Figure 255. Check In/Out Tab in Development Tools Options Window

Data sources user interface elements in the Development Tools Options window:

- **Server data source text box and change button.** ODBC data source of the server repository. When you click the Change button, the Change Data Source window appears (Figure 256) so you can enter and modify the ODBC data source parameters.

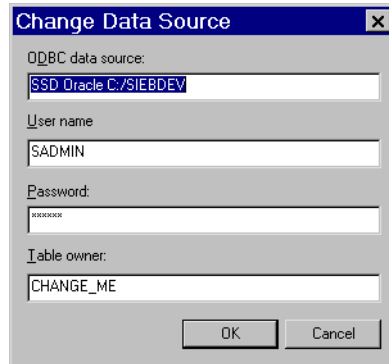


Figure 256. Change Data Source Window

User interface elements in the Change Data Source window:

- **ODBC data source.** Full ODBC data source string that provides communication with the server repository database.
- **User name.** User logon ID (in all uppercase) used to access the server database.
- **Password.** User password (in all uppercase) used to access the server database.
- **Table owner.** Table owner name used to access the repository on the server database.
- **Client data source text box and Change button.** ODBC data source of the local repository. When you click the Change button, the Change Data Source window appears so you can enter and modify the ODBC data source parameters. These parameters are similar to those described in the prior section for the server data source. In particular, the user name and password must be in all uppercase letters.

Determining Project Differences at Check-In Time

From the Project Differences window you can view details of changes made to checked-out projects prior to checking them in—it is a debugging tool that helps you find errors or omissions before your changes are committed to the server repository.

To invoke the Project Differences window

- Click the Diff button in the Check In dialog box (Figure 254 on page 1003). The Object Comparison window appears (Figure 257).

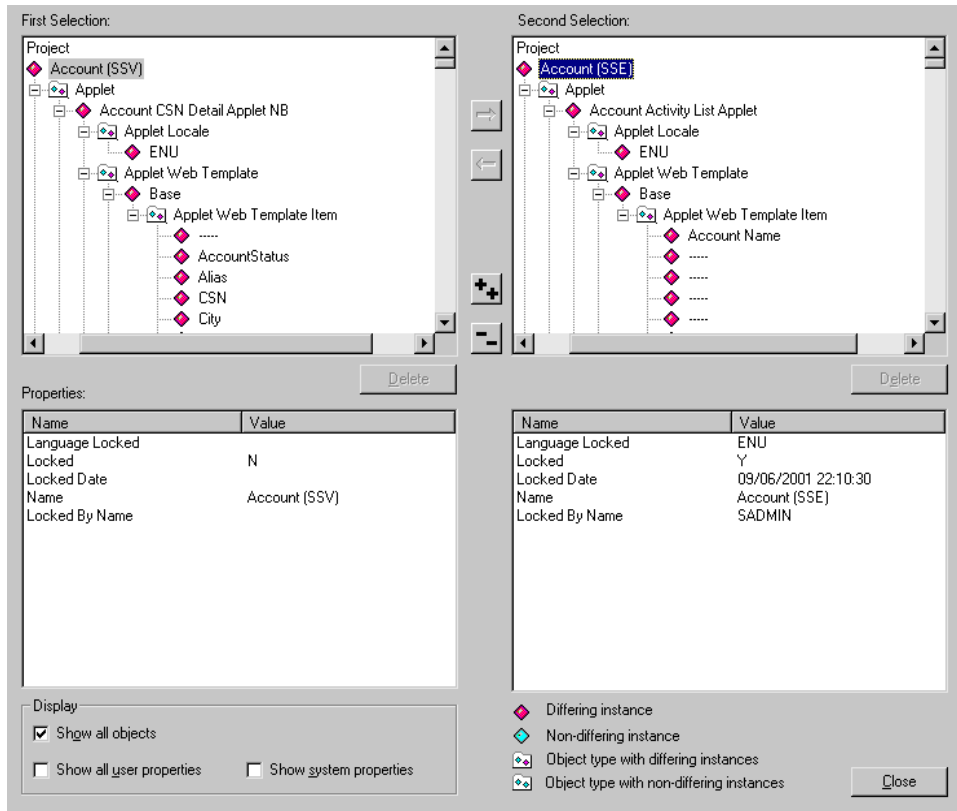


Figure 257. Object Comparison Window

The Object Comparison window compares the local project with the server project.

Project Differences Windowpane

The Project Differences windowpane displays the hierarchy of object definitions for which there are changes from the originally checked-out versions. It behaves like the Object Explorer in the Detail mode and is used for navigation. The hierarchy in this windowpane mirrors the object type/object definition hierarchy in a Siebel repository, but shows only changes since check-out rather than all repository or all checked-out object definitions.

Selection of an object definition in this hierarchy causes this object definition and the others at its level to appear in the Object Differences windowpane.

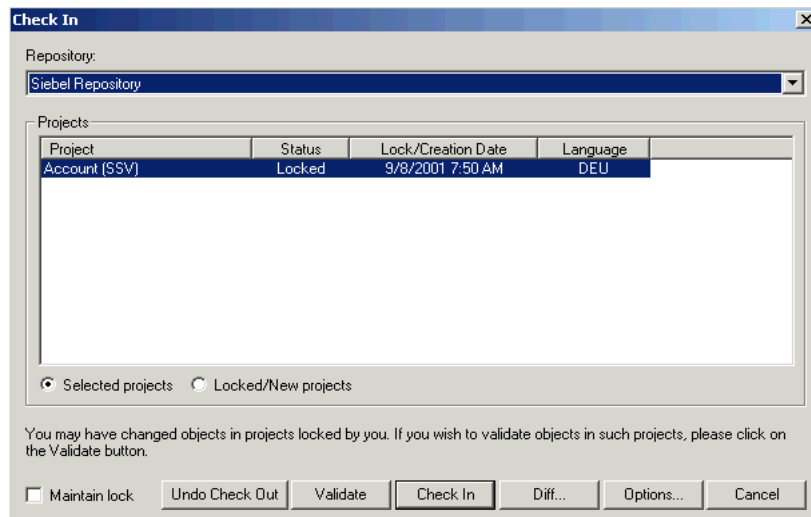
Undoing Check Out

You can undo projects that have been checked out from the Check In dialog box.

To undo a project check out

- 1** Choose Tools > Check In.

- From the Check In dialog box, select the project or projects for which you want to undo check-out, and click the Undo Check Out button.



The project is unlocked on the server but not on your local database.

If one of the projects you select is new, the Undo Check Out button is disabled.

NOTE: You can also use Get to overwrite a project that you have checked out from the server database. Perform the Get for the project you want to overwrite, and the project in your local database will be overwritten with the project from the server. Then check the project back in to the server. This will undo the lock for the project.

Locking Projects Directly

You can lock and unlock projects directly (as contrasted with the check-out procedure, which locks projects as it checks them out).

To lock local projects directly

- All Project object definitions have a Locked property that you can set to TRUE or FALSE in one of two ways:
 - Set the Locked property to TRUE in the Object List Editor by clicking the Locked field (if the Locked field has no check mark).
 - Set the Locked property to TRUE in the Properties window.

Figure 258 shows a locked project (Account).

Projects							
Name	Changed	Inactive	Locked	Locked By Name	Locked Date	Language Locked	Comments
Access Group							
Account	✓		✓	SADMIN	9/7/01 3:36:10 PM	ENU	
Account (SCW)	✓		✓	SADMIN	9/6/01 11:34:47 PM	ENU	
Account (SSE)	✓		✓	SADMIN	9/6/01 10:10:30 PM	ENU	
Account (SSV)							
Account (TAS)	✓	✓					
Account SAP - BW							
Acd							
Activity							
Activity (CC)							Activity - C
Activity (FS)							Field Engin
Activity (SCW)							
Activity (SSE)							
Activity (SSV)							
Activity Details (FS)							Field Engin

Figure 258. Locked Project (Account)

You lock at the project level, even though the Locked property is associated with a particular Business, Data, or User Interface Object. That is, when you lock a specific object, the entire project associated with the object is locked.

It might not seem logical to do a direct lock on a local project, because if it is not also locked on the server, other developers might check it out and your changes and theirs might ultimately conflict. Direct local locking *does* make sense, however, if you are:

- Prototyping your ideas and do not really want to prevent others from checking out the project you are working on
- Intending to discard your work when you are done

Be aware, though, that you cannot change your mind and decide to use your prototype created this way in your application, because:

- You can check in only those projects you have checked out.
- Your local project's definition will be overwritten the next time you get or check out that project.

Locking Server Projects Directly

Although you can do a direct lock on server projects (by changing the Locked property), Siebel Systems recommends that you never do this. Always lock by doing a check-out, instead, because:

- When you lock on the project directly, other developers who do a get on the object definitions in the directly locked project may find that the definitions are in an incomplete, inconsistent, and untested state.
- You, as the project owner, lose your ability to cancel the check-out and restore the original object definitions.

NOTE: You can check the project back in only in the project language in which it was checked out. If you switch language mode in Siebel Tools, you lose the lock you had with the project in the previous language mode.

Project Structure Considerations

The project structure supplied with Siebel Tools is usually well suited to having several developers work on the same repository without contention for the same object definitions.

To determine if the Siebel Tools project structure will work in your environment:

- Create an application development plan that includes a PERT chart showing dependencies and parallel activities.

- Analyze the plan to see if the project structure interferes with developers who need access to object definitions in the same projects at the same time. If so, break out groups of object definitions into separate projects to enable concurrent development.

During development, it is recommended that:

- You do not change the project structure in standard Siebel eBusiness Applications without a compelling reason.
- You limit application modifications as much as possible.

Application Development Projects

Project Structure Considerations

Configuring the Customer Dashboard

A

This chapter describes the tasks necessary to configure the Customer Dashboard. It also discusses the various methods for populating the customer dashboard, including populating it from the current record displayed, a communications event, Smart Script, and Search Center results.

Understanding the Customer Dashboard

The customer dashboard provides access to key customer information, such as contact name and account number, in an area of the screen that remains persistent as the user navigates in the application.

The customer dashboard is visible as a separate frame below the Communications Toolbar and screen tabs. It exists in addition to the primary content area of the Siebel Web client, also known as the base view, and the Search Center frame.

For more information about end user procedures, see *Siebel Call Center User Guide*.

How the Customer Dashboard is Populated With Data

The dashboard must be open to be populated. Once the dashboard is populated, the information will remain in the dashboard until the Clear Dashboard command is executed. During a session, all information populated in the Customer Dashboard is saved and the forward and backward buttons allow you to see the previous or next information populated in the dashboard. You can also configure a button on an applet to update the customer dashboard with information from the selected row. For more information, see [“Using Siebel VB Script and eScript” on page 1032](#).

The customer dashboard can be populated with data using any of the following methods:

- **Selected Record.** You can update the Customer Dashboard from a view by selecting a record and clicking the Update button in the Customer Dashboard. At this time it takes the primary business component for the view and updates the fields with data for this record.
- **Communications event.** When a user accepts an incoming call, the dashboard is automatically updated with contact information for the caller.
- **SmartScript answer.** You can configure the Customer Dashboard so that the answer to a question in a SmartScript automatically populates the dashboard.
- **Search Center results.** When the customer cannot be automatically identified from an inbound call, the user can search for the contact in Search Center and then click the Set Dashboard icon to populate the dashboard with the search results.

Architecture

The dashboard is implemented as a separate frame and view below the application toolbar and above the base view. It is based on a virtual business component called Persistent Customer Dashboard, which is associated with the Persistent Customer Dashboard business object. The objects in the Siebel repository that are related to the customer dashboard are:

- **Persistent Customer Dashboard business object.** Groups together business components that can populate the dashboard with data.
- **Persistent Customer Dashboard business component.** Is a virtual business component.
- **Persistent Customer Dashboard business service.** Controls the functionality of the dashboard.
- **Persistent Customer Dashboard applet.** Displays data in the user interface.
- **Persistent Customer Dashboard view.** Displays applet in the user interface.

The method used for updating the dashboard is UpdateDashboard. If you want to configure to update the dashboard from a button you would use the InvokeMethod function and pass a set of name-value pairs such as:

- Source Name: 'Base View'

- BusComp Name: 'Contact'
- RowId: 'srowid'

See [“Using Siebel VB Script and eScript” on page 1032](#) for more information.

Upon receiving the arguments, the member functions go through the set of fields configured to be displayed and populate the dashboard after retrieving the data from the database.

Predefined Behavior

The Customer Dashboard has been configured to display a set of fields from several business components, including Account, Contact, Employee, Service Request, Asset Mgmt, and so on. You can configure the Customer Dashboard to display information from other business components as well. See [“Configuring the Customer Dashboard” on page 1018](#) for more information.

NOTE: You can see which business components the Customer Dashboard is configured to display, by using Siebel Tools to review the list of Business Object Components associated with the Persistent Customer Dashboard business object. See [“Adding a Business Component to the Dashboard Business Object” on page 1019](#).

The Customer Dashboard can be populated with data from a single business component or multiple business components. However, the Customer Dashboard does not display data from multiple business components at one time. Rather, it is configured to display data in different contexts. For example, when the user is in the Accounts screen and clicks the Update button, account information is displayed; when the user is in the Contacts screen and clicks the Update button, contact information is displayed.

Enabling the Customer Dashboard

By default, the Customer Dashboard is enabled for Siebel Call Center, Siebel Sales, and Siebel Service. However, you can enable the Customer Dashboard for other applications as well.

To enable the customer dashboard

- 1 Find the Persistent Customer Dashboard business service.
- 2 Verify that the Inactivate property is set to FALSE.
FALSE is the default setting.
- 3 Add the target application as a value for the Applications user property.

For example, to activate Customer Dashboard for Siebel Employee Relationship Management, you would add Siebel ERM to the user property as shown in the table below:

Name	Value
Applications	Siebel Universal Agent; Siebel Field Service; Siebel Sales Enterprise; Siebel ERM

Configuring the Customer Dashboard

You can configure the customer dashboard to display data from any business component. The tasks for doing this include the following:

- [“Adding a Business Component to the Dashboard Business Object”](#)
- [“Adding a Business Component Lists to the Dashboard Business Service”](#)
- [“Mapping Business Component Fields to the Customer Dashboard” on page 1020](#)
- [“Creating Field Labels” on page 1022](#)
- [“Formatting Phone # Fields” on page 1023](#)
- [“Configuring the GoTo View Drop-Down List” on page 1024](#)
- [“Configuring Labels for GoTo Views” on page 1025](#)

Adding a Business Component to the Dashboard Business Object

You may need to display data from a business component that the Customer Dashboard is not preconfigured to support. To do this, first you must add the business component to the Persistent Customer Dashboard business object.

To add a new business component to the dashboard business object

- 1 In the Object List Editor, locate the Persistent Customer Dashboard business object.
- 2 Define a new Business Object Component (child of business object) for the business component.

Adding a Business Component Lists to the Dashboard Business Service

User properties of the Persistent Customer Dashboard business service specify the business components and the list of fields available to display in the Customer Dashboard. These user properties are also known as Business Component Lists. Each user property name begins with List and is appended with a value to make the name unique, for example List1, List2, and so on. The user property value identifies the name of a business component and the list of available fields.

For example, [Table 64](#) shows the preconfigured user properties that identify the Contact and Opportunity business components and their corresponding fields:

Table 64. Example Persistent Customer Dashboard Business Service User Properties

Name	Value
List1	Contact;Last Name;First Name;Full Name;Email Address;Work Phone #;Account;Account Location;Fax Phone #;Job Title;Mobile Phone #
List2	Opportunity;Name;Account;Account Location;Oppty Id;Close Date;Sales Rep;Revenue;Sales Stage

To define a user property to represent a business component and fields

- 1 In the Object List Editor, locate the Persistent Customer Dashboard business service.

- 2 Add the List user property and define the business component name and field names as the user property's value.
 - **Name.** The syntax for the name is the word List followed by a number. For example, List1, List2, List3, and so on.
 - **Value.** The value of the user property lists the name of the business component and then the corresponding field names. Each value must be separated by a semicolon. See [Table 64](#) for an example.

Mapping Business Component Fields to the Customer Dashboard

User properties defined for the Persistent Customer Dashboard business service map the available fields from the business component to fields on the Customer Dashboard applet.

The user property name identifies the Customer Dashboard fields, such as *Field 1*, *Field 2*, and so on.

NOTE: The following fields are preconfigured for the Customer Dashboard, Field 1, Field 2, Field 3, Field 4, Field 5, Field 10, Field 12. Field 4 is formatted to display phone numbers.

The value of the user property defines the business component list and one of the available fields. The syntax for the value is the name of the list user property, for example *List1*, followed by the position of the field in the list for that user property. For example, *List 1.1* is the first field available from *List1*, *List1.2* is the second field available from *List1*, and so on. For more information about business component lists, see [“Adding a Business Component Lists to the Dashboard Business Service” on page 1019](#).

For example, to display the Last Name field from the Contact business component (see List1 in [Table 65 on page 1021](#)) in Field 1 of the Customer Dashboard, you would define a user property as shown in [Table 65](#) below:

Table 65. Example User Property to Map Fields

Name	Value
Field 1	List1.1

You can display fields from more than one business component in a single Customer Dashboard field. To do this, you define multiple values for the Customer Dashboard field’s user property. For example, suppose that when the Customer Dashboard is in the context of Contacts, you want Field 1 of the dashboard to display Last Name. However, when the Customer Dashboard is in the context of Opportunities, you want Field 1 to display Opportunity Name.

To map multiple business component fields to a single customer dashboard field, you define the user property as shown in [Table 66](#). List 1.1. represents the first field of List one. List2.1 represents the first field of List2.

Table 66. Mapping Multiple Business Component Fields to a Single Dashboard Field

Name	Value
Field 1	List1.1;List2.1

The customer dashboard business service searches through the list of user properties, starting with Field, and looks for fields that are mapped to the dashboard from the current business component. For example, when the Contact business component is instantiated, the business service looks for Fields mapped from the Contact business component. Fields in the dashboard not mapped fields in the current business component remain empty.

Creating Field Labels

The field labels that appear in the Customer Dashboard are dynamic, they change depending on the data being displayed in the dashboard. When no data is available for the dashboard, the labels for the default business component are displayed. The default business component is specified in the Customer Dashboard business service. Contacts has been preconfigured as the default business component for the dashboard.

There are placeholder controls, such as Label 1, Label 2, and so on, that are predefined in the Siebel Repository. There are also predefined business service user properties, also named Label 1, Label 2, and so on, that map these placeholder labels to fields on the Customer Dashboard.

If you add additional fields to the Customer Dashboard, you define the actual labels that will replace placeholder labels at runtime. You define the actual labels by creating an additional applet control for each business component field that you want to display. The naming convention for the applet control identifies it as a Label, and identifies the business component and field that determine when it should be displayed.

To create an applet control to represent a label

- 1** Go to the Persistent Customer Dashboard applet.
- 2** Create a new applet control.
- 3** In the Name field, enter a name using the word Label, followed by a space, then followed by the business component name and field name separated by a dot.

For example: Label ServiceRequest.SR Number

- 4** In the Caption field, specify the text that you want to appear in the Customer Dashboard.
- 5** Repeat this process for each label that you want to display for a particular dashboard field.

Formatting Phone # Fields

You can configure the Customer Dashboard to recognize different telephone extensions. You use a business service user property to define the parameters that associates your company's telephone switch extensions to their full-length phone numbers. The user property name is Phone Number Prefix. Three values, separated by semicolons, define the parameters. The values are as follows:

- The first value specifies the number of digits in an extension.
- The second value specifies the number of digits to remove from the front of the extension.
- The third value specifies the prefix to append to the beginning of the number.

Consider the example shown in [Table 67](#):

Table 67. Phone Number Prefix Example

Name	Value
Phone Number Prefix 1	5;1;650555

The example shown in [Table 67](#) would allow a user to dial the extension 24565. The extension has 5 digits. The first digit, 2, is removed. The prefix, 650555, is added. The resulting phone number is 650-555-4565.

Configuring the GoTo View Drop-Down List

The Customer Dashboard includes a drop-down list that allow users to navigate to additional views related to the current record. The list of views changes depending on the data currently displayed in the Customer Dashboard.

The list of views available in the Go To drop-down list are configured using business service user properties, such as View 1, View 2, and so on. At runtime the Persistent Customer Dashboard business service searches through the list of user properties that start with View, finds the display name for the associated view, and then adds the name to the Go To drop-down list.

NOTE: The name of the view specified in the user property must exactly match the name as it is defined in the Siebel repository.

You can modify the views associated with the preconfigured View user properties or add additional views. The syntax for the user property is:

- **Name.** The word View followed by a number.
- **Value.** The value of the View user properties includes the following values separated by a semicolon:
 - Name of the business component
 - Name of the view
 - Name of the primary applet on the view
 - Name of the foreign key on a linked business component. This value is only necessary when you are navigating to a view based on a business component other than the current business component of the dashboard.

For example, if the dashboard is configured to display data from the Contact business component, and the All Activities view is a view listed on the GoTo drop-down list, this value would specify the foreign key in the Action business component that points back to Contacts. It allows a query of all activities related to the contact currently displayed in the dashboard.

For example, [Table 68](#) shows several preconfigured View user properties.

Table 68. Example View User Properties

Name	Value
View 1	Contact; All Activity List View; Activity List Applet With Navigation; Contact Id
View 2	Contact; Contact Activity Plan; Contact Form Applet
View 3	Contact; Agreement List View; Agreement List Applet No Parent; Contact Person Id

NOTE: In View 1 shown in [Table 68](#) specifies that when the dashboard is populated with data from the Contact business component, the All Activities view should appear in the GoTo View drop-down list. After the user selects the view from the drop-down list, only records for the current Contact ID appear in the view.

Configuring Labels for GoTo Views

You can specify the view labels for each view that you configure to appear in the GoTo drop-down list.

To configure labels for GoTo views

- 1** Locate the Persistent Customer Dashboard applet.
- 2** Go to the Applet Controls.
- 3** Create a new control for the view that you have configured to appear on the GoTo drop-down list.

The name of the control must be in the format of the word Label followed by the name of the View user property. For example, Label View 1 for View 1, and so on.

- 4** In the Caption property, enter a text you want to appear in GoTo drop-down list.

Modifying the Look and Feel of the Customer Dashboard

You can modify the look and feel of the Customer Dashboard, including its color, size, and location.

Changing the Background Color and Border

You can change the background color and border color properties in the `main.css` file, which is located in `PUBLIC\Language_Code\FILES\` directory of your Siebel installation, where language code represents the three-letter code for your installation's language pack.

To modify the background color or border

- 1 Locate the `main.css` file in the `PUBLIC\Language_Code\FILES\` directory of your Siebel installation.
- 2 Open the file with Notepad or other editor.
- 3 Find the following section and modify the values for `dashbrdBorder` and `dashbrdBack` as necessary.

```
/*-----*/  
/*Dashboard Definitions*/  
/*-----*/  
  
.dashbrdBorder {background-color:#6666CC;}  
  
.dashbrdBack {background-color:#E0E0E0;}
```

Changing the Size and Location

You can change the size and location of the dashboard. For example you can make the dashboard appear in the bottom of the view or you can make it occupy the complete horizontal space or a certain percentage of the content frame size.

NOTE: The dashboard is located outside of the Content Frame and you can only move the dashboard around the Content Frame of the application.

All changes can be done by modifying five template files:

- CCFrameContent_V.swt
- CCFrameContent_VS.swt
- CCFrameContent_VD.swt
- CCFrameContent_VSD.swt
- CCAppletDashboard.swt

A move to the bottom of the view is a mirroring operation. A move to the left or right of the content frame will have to take into consideration the sizing issues present when search center is open. Moving the dashboard to the right side is not recommended because it breaks the connection between actions taken in Search Center and results returned in the main content area. When making changes to the dashboard location, be sure to test that the dashboard frame, content frame, and search center frame are working properly together.

Configuring Communication Events

One of the ways to populate the Customer Dashboard is from communication events, such as an inbound email message, voice call, or web collaboration work item. The *Multichannel Def A* configuration has been preconfigured to populate the Customer Dashboard with contact information for certain communication events. However, you can configure any communication event to populate the customer dashboard for any business component, based on information passed to the event.

The API for the communication between communication events and the customer dashboard is a member function `UpdatefromCTI` of the Customer Dashboard business service. The CTI administration views are preconfigured to call `InvokeMethod_` (with `UpdateDashboard` as a parameter) when a significant event occurs and pass variables, such as Phone number and Number of calls in queue, as arguments.

To populate the customer dashboard during a communications command or event, you need to call the method to update the customer dashboard and pass three parameters, including the business component, the field for that business component, and the value that you are getting from this communication.

For example, the parameters listed in [Table 69](#) instruct the customer dashboard to populate with contact information for the contact whose Work Phone # matches the ANI of the inbound call.

Table 69. Customer Dashboard Parameters for Communications Events

Parameter	Example Value
<code>ServiceMethod</code>	Persistent Customer Dashboard.Update Dashboard from CTI
<code>ServiceParam.Field</code>	Work Phone #
<code>ServiceParam.Value</code>	{ANI}
<code>ServiceParam.BusCompName</code>	Contact

You can also call the customer dashboard business service from the communications event log. See the steps below for an example.

- 1 Locate the Event Handler `InboundCallReceived`.
- 2 Click on the Associated Event Logs tab.
- 3 Drill down on the log `LogIncomingCallContactFound`.

4 In the log parameters, you define the following parameters:

Parameter	Example Value
ServiceMethod	Persistent Customer Dashboard.Update Dashboard from CTI
ServiceParam.Field	Id
ServiceParam.Value	{Contact.Id}
WorkTrackingObj.ContactId	{Contact.Id}

Configuring SmartScripts

You can configure the Customer Dashboard so that the answer to a question in a SmartScript automatically populates the dashboard. The configuration tasks required to do this are:

- [“Activating the SmartScript Player”](#)
- [“Mapping SmartScript Variables to Customer Dashboard Fields”](#)
- [“Configuring SmartScripts to Save Answers”](#)

NOTE: You cannot update the Customer Dashboard from VB or eScript that executes within a SmartScript. There is a one-to-one relationship between a user interface event and the ability to update a frame in the application. Because each user interface event within a SmartScript updates the SmartScript frame, it cannot also update the Customer Dashboard frame. If you were to pass parameters to the Customer Dashboard from VB or eScript within a SmartScript, the dashboard would receive the parameters but would not be able to display them.

Activating the SmartScript Player

When using the customer dashboard with SmartScripts you need to verify that the Notify Dashboard property of the Smart Script Player Applet (Tree Only) is set to true.

To verify that the Notify Dashboard user property is set to TRUE

- 1 Locate the applet named Smart Script Player Applet (Tree Only).
- 2 Locate the applet user property named Notify Dashboard.
- 3 Verify that the value of the user property is Y.

Mapping SmartScript Variables to Customer Dashboard Fields

You must map the variables in the SmartScript to fields on the Customer Dashboard. You do this by defining the SmartScript List user property of the Persistent Customer Dashboard business service. The mechanism for doing this is similar to defining user properties for a business component list. See [“Mapping Business Component Fields to the Customer Dashboard” on page 1020](#).

The user property name is *SmartScript List*. The value for the user property specifies the variables from SmartScript answers that are to be displayed in the Configuration Dashboard.

To define the SmartScript List user property

- 1 Locate the Persistent Customer Service Dashboard business service.
- 2 Define a user property with the name SmartScript List and values that represent the variables from the SmartScript.

Name	Value
SmartScript List	Fname;Lname;Phone;Interests

NOTE: The values of the user property must exactly match the variable names specified in the SmartScript.

Configuring SmartScripts to Save Answers

To be able to pass SmartScript answers to the Customer Dashboard, you must configure the SmartScript to save the answers. You do this in the SmartScript Administration view.

To configure SmartScripts questions

- 1** Navigate to Site Map > SmartScript Administration > Questions.
- 2** In the Questions list, select a question.
- 3** In the More Info form, enter the name of the variable in the Save User Parameters field.

This enables the answer to be saved as a global variable to the script.

NOTE: The name of the variable must exactly match the name as it is listed in the SmartScript List user property of the Persistent Customer Dashboard business service. See [“Mapping Business Component Fields to the Customer Dashboard” on page 1020](#).

- 4** From the Show drop-down list, choose Scripts.
- 5** In the Translation form, enter the name of the variables from each question.
The syntax for entering the variables is the name of the variable enclosed in brackets, separated by spaces; for example, [Fname] [Lnmae] and so on.
- 6** Repeat the steps above for each question you need to configure for the customer dashboard.

The values for the variables in the Dashboard Text field are passed to the Customer Dashboard.

Using Siebel VB Script and eScript

The customer dashboard provides APIs to pull information from or push information to the dashboard using Siebel VB or eScript. Since the customer dashboard is a separate frame in the application, it requires a UI event to update the dashboard using Siebel VB or eScript. This means that you can only update the dashboard in this manner by adding a button to an applet and then calling the Update Dashboard command. When adding the button make sure that you set the Target Frame View property to Dashboard.

NOTE: The Customer Dashboard architecture only allows one UI update for each user UI event. For example, if you put a button on a view, the clicking of the button is one UI event. For that event, you can only execute one UI update, such as updating the dashboard. The code behind a single button cannot have two UI updates, such as updating the dashboard and then going to a new view in the main frame of the application.

To add a button that calls the Update Dashboard Command

- 1** Locate the applet on which you want to place the button.
- 2** Add the button as an applet control and the script behind the button
See [“Customer Dashboard Commands”](#) for a list of available commands.
- 3** Set the button property, Target Frame View to Dashboard.

Customer Dashboard Commands

Since the customer dashboard is a business service, in your script you will need to use the GetService ("Persistent Customer Dashboard") command. Using the code behind a button you may either push information to the dashboard or pull information from the dashboard.

There are two commands to pull information from the dashboard including getting the record id for the current record populated in the dashboard or obtaining individual field values for fields populated in the dashboard. Details for these commands are described in the following sections.

GetCurrentContactId

This command returns the record ID for the current record populated in the dashboard. For example, if the record is from the Contact business component, then it returns the ContactId; if the record is from the Account business component, then it returns the AccountId.

No input argument should be specified.

The output argument is always "ContactId."

NOTE: The "ContactId" is a variable used by the dashboard but this refers to the record ID for whichever business component is populated in the dashboard.

For example:

```
bs.InvokeMethod("GetCurrentContactId",inargs,outargs); var  
fvalue = outargs.GetProperty("Contact Id");
```

GetDashboardFieldValue

This command returns the current field value for the record populated in the dashboard. The input argument is the name-value pair for the dashboard field. The output argument is "Field Value."

For example:

```
inargs.SetProperty("Field Name","Field 4");  
bs.InvokeMethod("GetDashboardFieldValue",inargs,outargs); var  
fvalue = outargs.GetProperty("Field Value");
```

Update Dashboard

This command is used to populate the dashboard with a new record.

- Source Name: Base View
- Buscomp Name: Contact
- RowId: E301

For example:

```
inpargs.SetProperty("Source Name","Base View", "Buscomp Name",  
"Contact", "RowId", "E301"); bs.InvokeMethod("Update  
Dashboard",inpargs,outargs);No output argument
```

NOTE: In Siebel 7 version 7.0.3 and 7.0.4 there are two spaces between "Buscomp" and "Name" in the second parameter. In subsequent versions, there is one space.

Siebel eScript Example

The example below is a script, written in Siebel eScript, that uses the dashboard commands. It gets the contact ID, Field 4, and Field Time for the current record populated in the dashboard and prints those values to a file.

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

```
function Script_Open ()  
{  
    var fnl=Clib.fopen("d:\\sabari5.txt", "wt");  
    var bs = TheApplication().GetService("Persistent Customer  
dashboard");  
    var inpargs= TheApplication().NewPropertySet();  
    var outargs = TheApplication().NewPropertySet();  
  
    bs.InvokeMethod("GetCurrentContactId",inpargs,outargs);  
    var fvalue = outargs.GetProperty("Contact Id");  
    Clib.fprintf (fnl, "The current id in the dashboard = %s  
\n",fvalue);  
  
    inpargs.SetProperty("Field Name","Field 4");  
    bs.InvokeMethod("GetDashboardFieldValue",inpargs,outargs);  
    var fvalue = outargs.GetProperty("Field Value");  
    Clib.fprintf (fnl, "The Account Name in the dashboard = %s  
\n",fvalue);  
  
    inpargs.SetProperty("Field Name","Field Time");  
    bs.InvokeMethod("GetDashboardFieldValue",inpargs,outargs);  
    var fvalue = outargs.GetProperty("Field Value");  
    Clib.fprintf (fnl, "The current time of the agent/customer in the  
dashboard = %s \n",fvalue);  
}
```

```
    Clib.fclose(fn1);  
    return(ContinueOperation);  
}
```

Siebel VB Example

Below is an example script written in Siebel VB that uses the dashboard commands. It gets the contact ID, Field 4, and Field Time for the current record populated in the dashboard and prints those values to a file.

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

```
Sub Script_Open  
  
    Dim bs as Service  
    Dim inpargs as PropertySet  
    Dim outargs as PropertySet  
    Dim fvalue as String  
  
    Open "d:\sabari.txt" for Output as #1  
    Set bs = TheApplication().GetService("Persistent Customer  
dashboard")  
    Set inpargs = TheApplication.NewPropertySet  
    Set outargs = TheApplication.NewPropertySet  
  
    bs.InvokeMethod "GetCurrentContactId",inpargs,outargs  
    fvalue = outargs.GetProperty("Contact Id")  
    Write #1, "The current id in the dashboard = " & fvalue  
  
    Inpargs.SetProperty "Field Name","Field 4"  
    bs.InvokeMethod "GetDashboardFieldValue",inpargs,outargs  
    fvalue = outargs.GetProperty("Field Value")  
    Write #1," The Account Name in the dashboard = "& fvalue  
    Close #1  
  
End Sub
```

About Dual Personalization

The Personalization engine has the ability to personalize the Call Center application based on both the agent's profile and the customer's profile. The agent's profile is loaded when the agent logs into the Call Center application. The customer's profile is loaded when the customer information is populated in the Customer Dashboard. This allows the agent to see customer-specific information based on the personalization rules created by your Siebel administrator.

For example, based on the customer's profile you could show a different applet or view to the agent. You could have a Recommended Products applet which only shows products for this customer based on products he previously purchased.

To access the profile information you create personalization rules. The Me.attribute allows you to access agent's profile information and the You.attribute to access the customer's profile information. Examples of these commands are displayed below.

- `GetProfileAttr("You.Last Name");`
- `GetProfileAttr("Me.Last Name");`

See *Personalization Administration Guide* for more information about profile attributes and creating personalization rules.

Index

Symbols

- .sif file. *See* archive file
- .spf file. *See* patch file
- .srf file
 - note, about compiling projects 277

Numerics

- 2dBar bar chart, about and screen
 - example 727
- 2dHorizBar bar chart, about and screen
 - example 728
- 2dLine line chart, about and screen
 - example 730
- 2dScatter scatter chart, about and screen
 - example 735
- 2dSpline line chart, about and screen
 - example 731
- 2dStackedBar bar chart, about and screen
 - example 729
- 3dBar bar chart, about and screen
 - example 722
- 3dHorizBar bar chart, about and screen
 - example 724
- 3dLine line chart, about and screen
 - example 730
- 3dPie pie chart, about and screen
 - example 734
- 3dSpline line chart, about and screen
 - example 732
- 3dStackedBar bar chart. about and screen
 - example 726

A

- About Repository File window, getting information on 990, 991

- access control
 - business components, about configuring 238
 - organization-related data 242
 - party business components 245
 - party described 240
 - person-related data 240
 - S_PARTY table 246
 - summary 248
- Account Entry applet, property settings 490
- Account List applet, property settings 490
- action arguments, enabling 678
- Active X controls
 - about 776
 - applet, adding to 778
 - distributing, about 783
 - methods, viewing list of available methods 782
 - properties, changing in the native property sheet 780
 - properties, changing in the Properties window 780
- Add button, association applets, in 710
- Advanced Database Extensibility
 - See also* database extension, planning and design; database extension, implementing
 - about using to extend data model 251
 - note, changes unupgradable 292
 - Table Wizard, columns created by 297
 - Table Wizard, note, types of tables created 293
 - Table Wizard, using 293
 - tables, types 292

- Alias property, about 196
- AND operator, search specification, in
 - a 349, 486
- Applet Designer 92
- Applet Menu Wizard
 - new method menu item, creating 592
 - using 591
- Applet Method Menu Item Object, using to
 - create applet menu 593
- Applet Method Menu Item Wizard, about
 - implementing custom menu options 509
- applet object definition, about 474
- Applet object type
 - Tree applet properties 761
- Applet property
 - Tree Node object type 761
- Applet template, template type 793
- Applet Toggle picklist, location and use
 - of 809
- Applet Web Template
 - object properties 828, 829
 - Web-related object that contains the list items, applet controls, and Web controls 510
- Applet Web template
 - described 73
- Applet Web template item
 - described 73
- Applet Web template item, user interface
 - object 452
- Applet Web Template Items object, applet
 - object definition
 - form applet 464
 - list applet 476
- Applet Web Template Layout Window
 - tree applets, using and options 764
- Applet Web Template object, applet object
 - definition
 - form applet 464
 - list applet 475
- Applet Web template, user interface
 - type 452
- applet-level menus
 - invoking, about and screen example 865
- applets
 - See also* list applets; form applet; pick applets; specialized applets; tree applets; user interface objects, configuring
 - Account List View, property settings, table of 490
 - Active X controls, adding to 778
 - applet menus, creating using wizards 593
 - applet modes 793
 - Applets window, deleting from 576
 - association applet, about and example 459
 - bitmap button controls, about using
 - in 584
 - catalog style list applets 862
 - chart applet, about and example 458
 - configuring applets using wizards 623, 626
 - Controls/Columns window, about using
 - to edit applet layout 110
 - defined 453
 - docking, preventing 113
 - docking/undocking 113
 - explorer applet, about and example 460
 - file attachment applet, about and example 461
 - form applet, about and example 454
 - form applets, about using persistently editable forms 832
 - hiding 113
 - language-independent code, creating for
 - Workflow Policy Column 677
 - language-independent code, creating for
 - Workflow Policy Program Argument 679
 - links, about using in multi-value group applet 399
 - list applet, about and example 455
 - list applets, physical ui navigation 833

- method menu, modifying using the
 - Applet Menu Wizard 591, 592
- multi-value group, about and
 - example 456
- multi-value group, about using links to
 - implement 394
- new method menu item, creating using
 - the Applet Menu Wizard 592, 593
- object types, described 73
- pick applets, about and example 456
- popups, about creating in SWE 772, 849
- Search Specification property, about,
 - sample search expressions, and
 - syntax 486 to 488
- template used to layout fields and
 - controls 793
- toggle applets, about and creating 849
- toggle applets, coding guidelines 675
- tree applet, about and example 460
- tree applets 853, 860
- user interface element, described and
 - example 69
- user interface object 452
- Web layout, adding to Web layout of the
 - view 566
- Web layout, editing using the Web Layout
 - Editor 547
- Web-related object that maps applets to
 - View Template 510
- Web-related object, script exposed in a
 - Siebel Web application 510
- Applets by BusComp report,
 - described 144
- Applets window
 - deleting applets from window 576
 - described 109
- applets, modifying appearance
 - conditional mapping, removing 888
 - conditional mappings, about 887
 - controls/list columns, displaying or
 - removing at runtime 890
 - More button, using to expand applet 891
- application
 - defining, about 576
 - naming conventions for 577
 - page tabs, creating in 577
 - screen menu items, associating screens
 - to 578
- application definition
 - creating the Web application definition in
 - Siebel Tools 168
 - overview of the creation process 166
 - strategies to optimize application
 - performance 169
- application executables, confusion with
 - applications user interface
 - element 72
- Application Find object type 341
- application launching routines 173
- application object definition, about and
 - contents of 508
- application object definition, described 72
- Application Script object type, about using
 - to implement scripts 508
- Application Upgrader. *See* Siebel
 - Application Upgrader
- application window
 - Siebel Tools 91
- application window, Siebel Tools 90
- application, user interface object type 451
- applications
 - logical UI object types, described 72
 - user interface element, described and
 - example 72
- applications, configuring
 - associating each applet and view with the
 - correct template 166
 - compiling the repository changes into an
 - .SRF file 169
 - creating applets and views 166
 - deploying the application 171
 - establishing mapping between control
 - and templates 168
 - modifying templates to create corporate
 - image 168

- overview of the configuration process 161
- personalizing your Web application, key points 185
- relationships between components in a Siebel Web application (diagram) 162
- testing the application 170
- architectural diagrams
 - of user interface object types 338
- architecture
 - Business Objects layer 62
 - class, defined 54
 - configuration guidelines 163
 - Data Object layer, described 58
 - data object types, described 62
 - high interactivity, about 80
 - layered structure of 49
 - object definitions and properties, defined 51
 - object definitions, value types 117
 - Physical User Interface layer, described 74
 - repository, defined, Contact object type example, and contents 54, 56
 - servers, components deployed on 77
 - User Interface Objects layer, diagram example and described 68
- archive file, contents of 918
- Assignment Criteria and Skills, configuring for MLOV 681
- Assignment Manager
 - about using 187
- association applet (Contact Assoc applet)
 - object type, about 710
- association applet (Industry Assoc applet)
 - object type, about 715
- association applets
 - about and example 459
 - about and screen example 702
 - architecture (diagram) 708
 - filtering or constrained, about 715
 - master-detail view, about invoked from, about and screen example 707
 - multi-value group applets, about invoked from, about and screen examples 711
 - multi-value group applets, object definitions (diagram) 714
 - multi-value group applets, object definitions, list of 714
 - note, filtering, about 706
 - object definitions, list of 709
 - object relationships (diagram) 709
 - row relationships in a many-to-many relationship, described and diagram 705
 - row relationships in a one-to-many relationship, described and diagram 704
- attachment applets, about using for attachment support and synchronization 768, 769
- attachment business component, requirements and fields 769, 771
- attribute column, about 228
- Attribute Differences windowpane, about 929, 931
- Attribute Mapping, about 233
- attributes, locale-specific
 - exporting 605
 - importing 605, 607
- ATTRIBxx extension columns, about and example 361
- Auto Primary property, about 415
- axis
 - chart applet axis terminology 720
 - X axis or Z axis labels, limiting and sorting 749
- AxisLabel value, Chart Element object type 750
- AxisLineGrid value, Chart Element object type 750
- AxisTitle value, Chart Element object type 750

B

- bar charts
 - 2dBar, about and screen example 727
 - 2dHorizBar, about and screen example 728
 - 2dStackedBar, about and screen example 729
 - 3dBar, about and screen example 722
 - 3dHorizBar, about and screen example 724
 - 3dStackedBar, about and screen example 726
- base columns, about 218
- Base mode for applets, described 793
- Base Table property Table, about 196
- base tables
 - business component, of 344
 - columns, length limit to add to base tables 373
 - custom extension columns, described 61
 - extension column data, displaying 278
 - multi-value link object definition 410
 - note, modifying user keys 227, 232
 - two different contexts used in 194
- BC Read Only Field user property
 - about, properties, and example 352
- bitmap button controls
 - applets and views, about using in 584
- Bitmap Category object type
 - associating bitmap images, about 582
- Bitmap objects
 - images, configuring as 892, 893
 - images, previewing 114
 - LGIF and JPG formats, about using 893
- Bookmarks windows
 - described 112
 - docking, preventing 113
 - docking/undocking 113
 - hiding 113
- Boolean columns, Column object type used for 216
- browser
 - browser group specific templates, designing 884, 887
- browsers, testing phase of configuration process 170
- BusComp property, Business Object component, about 425
- business component
 - See also* virtual business components
 - business object, described and example 64
 - business object, reuse described and example 65
 - described and examples 62
 - object type described 66
- Business Component and Fields report, described 144
- Business Component object type, about 382
- Business Component property
 - Applet object type 761
 - Tree Node object type 762
- business component property
 - Single Value Field object type. about 534
- Business Component Wizard, using to create Business Component objects 444
- business components
 - See also* originating business components
 - about 342
 - association applets, (Opportunity and Contact) object type 710
 - attachment business components, requirements and fields 769, 771
 - base table, of 344
 - base table, two different contexts used in 194
 - Business Component Wizard, using to create business components 444
 - business objects, mapping to 441
 - chart applets, mapping 737, 740
 - chart applets, properties 740, 741
 - copying, guidelines for 973

Details Visualization View, example and described 146
 Details Visualization View, generating for 148
 external business components 360
 field, adding to 433
 field/column relationships, about 343
 intersection business component 356
 joined table and extension tables 345
 map to underlying tables, viewing 145
 MLOV implementation 674
 multi-value applets, originating business component object type 691
 multi-value group, about storing data and identifying primary record 695
 pick applet, Picklist Generic business component, about and examples 647
 properties, described 432
 read-only behavior, configuring 350
 Relationships Visualization View, example and described 149, 150
 Search Specification property, about, sample search expressions, and syntax 348
 Single Value field properties 434
 sort specification property, about, example, and syntax 347
 static picklist, Picklist Generic business component object type 642
 View Details diagram, displaying for selected business component 150
 View Relationships, navigating with the business components version of 150
 viewing link using multi-value link object definitions 145
 virtual business components 358
 business object
 about 62
 Applet window, about using to display applets 109
 Bookmarks window, about using to display Bookmarks 112
 Business Object layer, described and example 64
 Business Object layer, reuse described and example 65
 component object definition, described 66
 Controls/Columns window, about using to edit applets 110
 Details Visualization View, example and described 147, 148
 master-detail view object definition 493
 object type, described 66
 Web Templates window, about using to display Web templates 111
 Business Object and Components report, described 144
 Business Object Component object type, about 424
 Business Object Component, about 209
 business object components
 master-detail view object definition 493
 Business Object object type, about 424
 business object types
 note, Business Object, not confused with 66
 business objects
 about 418
 architecture (diagram) 423
 intersection tables, use with 209
 logical database diagram and description of 418, 421
 object definitions, list of 424
 object definitions, relationship between (diagram) 424
 Business Objects layer
 business components, business object described and example 64
 business components, business object reuse, described and example 65
 business components, described and examples 62
 business objects, defining 434

- Business Component child object types, viewing 435
 - business component properties, described 432
 - business component, adding a field to 433
 - business components, mapping to 441
 - Business Object object types, key properties of 440
 - development sequence for defining 430
 - sequence order for defining 427
 - user interface object and business objects, mapping required (diagram) 428
 - business services
 - defined and overview 425
 - described 67
 - BusObject Designer 91
 - BusObject Repository Manager 98
 - buttons
 - control properties 469
 - By picklist
 - about 743
 - configuring, properties and examples 746
- C**
- calculated fields
 - about 367
 - data types, specifying 368
 - calculator
 - ellipsis icon, providing for 468
 - list applet type property 485
 - Calendar business component, note, about using in a link 397
 - calendar/time control, list applet type property 485
 - calendars
 - ellipsis icon, providing for 468
 - list applet type property 485
 - Cancel button, in association applets 710
 - Caption property
 - Control object type, about and example 466
 - form applet control property 535
 - cascade copy, about constructing with multi-value link 417
 - Cascade Delete property
 - about and values 399
 - Cascading Style Sheets
 - examples of use 915
 - Cascading Style Sheets, described 74
 - catalog style list applets
 - about and screen example 862
 - configuring 862
 - example 863
 - Category Field property
 - By picklist, populating 747
 - Chart object 740
 - Char, note, defining column as 216
 - Character data type, Physical Type property of Column object type 216
 - Chart Applet Wizard, using to create a new chart applet 751
 - chart applets
 - about and example 458
 - about and screen example 718
 - axis example (diagram) 721
 - axis terminology, table of terms 720
 - bar charts, types of and screen examples 722
 - chart layout options, about 722
 - line charts, types of and screen examples 729
 - new chart applet, using Chart Applet Wizard to create 751, 755
 - pie charts, types of and screen examples 733
 - scatter charts, types of and screen examples 735
 - chart applets, configuring
 - about 737
 - business component mapping, about and screen examples 737
 - business component properties 740

- By picklist, properties and examples 746, 747
- chart element object type, about and types 750
- ComboBox controls, control type for picklist 744
- multiple line graphs plotted against one Y axis, about and object properties 748
- performance considerations 751
- picklists, configuring 742
- second By picklist, properties and examples 747
- Show picklist, properties and examples 744, 746
- size 751
- two Y axes charts. configuring and properties 749
- X axis or Z axis labels, limiting and sorting 749
- Chart Elements, about and types 750
- check box controls, about and example 470
- Check button, in association applets 710
- Check In dialog box, user interface elements, described 1003, 1004
- check mark
 - See also* date, and time setting; object definitions
 - list column, about displaying in 473
 - meaning of 135
- Check No Match Property, using with a primary join 416
- Check Out dialog box, user interface elements described 996, 998
- checking in
 - project differences, viewing 981, 982
 - repositories, integrating with external source code control 957
 - source control integration, Check In/Out tab, about and options 965
- checking in projects
- Check In dialog box, user interface elements 1003
- data source options 1005
- guidelines 1004
- object definitions, new or modified 1001
- object definitions, viewing hierarchy changes 1009
- checking out
 - project differences, viewing 981, 982
 - repositories, integrating with external source code control 957, 965
 - source control integration, Check In/Out tab, about and options 965, 966
- checking out projects
 - Check Out dialog box, user interface elements, described 996, 998
 - checking out copy 995
 - data source options 1005, 1007
 - undoing project check out 1009
- check-mark icon, about attaching to text box 467
- Child Business Component property
 - Link object type 397
- Class property
 - Applet object type 760
 - Business Component object type 432
 - Single Value Field object type 534
- class, defined 54
- CLEAR value, Cascade Delete property 399
- coding guidelines for MLOV 675
- Color, Chart Element object type 750
- Column Name property, about 226
- column object definition, described 62
- Column objects
 - about and styles 215
 - column object types, list of 216
 - data columns, about 218
 - extension columns 218
 - system columns 222
- Column property, Single Value Field object type 434
- columns

Char, note, defining column as 216
 custom columns, adding to the database 257
 database, about configuring EIM and docking/routing interfaces 228
 extension column names, about and examples 261
 intersection data 212
 note, length limit to add to base tables 373
 combo box controls, about and example 470
 Combo line chart, about and screen example 733
 ComboBox value, about implemented as a combo box control 470
 Command object
 picklist, valid values 586
 properties, using the Command Object Wizard to select 586, 590
 Command Object Wizard, using to select object properties 586
 Comment property, Dock Object Visibility Rule object type 316
 comparison operators
 search specification, in a 349, 486
 compiling projects
 See also projects
 caution, about compiling or modifying .srf file 130
 object compiler, accessing 130
 single or group objects, compiling 131
 compound queries, about creating and table of 139
 conditional mapping
 applets, about using to modify applets 887
 applets, displaying or removing control/list columns at runtime 890, 891
 applets, using More button to expand applet display 891
 removing 888
 configuration
 MLOV configuration compatibility 674
 MLOV implementation 674
 multilingual target columns, enabling 663, 664
 Siebel Anywhere for MLOV-enabled fields 683
 Siebel Assignment Manager for MLOV-enabled fields 681
 Siebel Field Service for MLOV-enabled fields 683
 target columns for multilingual use 663
 visibility rules and MLOV 658
 Workflow Manager for MLOV 676
 configuration goals and objectives 160
 configuration guidelines
 architectural considerations 163
 configuration tasks 164
 conflict resolution
 individual object definitions, about adjusting for (screen diagram) 927
 object definitions, displaying hierarchy of differences, about 927
 object definitions, displaying property value conflicts for selected definitions 929, 931
 objects definitions, displaying one to a row, about 928
 Conflicting Objects windowpane, about 927, 928
 Constrain property, about using to filter pick applets 631
 Contact Account applet, about 491
 Contact Form applet, about 491
 content area, showing additional views 824
 Contract All option, Visualization view shortcut menu 148
 control column
 field values, rendering as icons 894, 896
 images, using as links in 897
 Control object definition
 applet object definition 475
 described 73

- user interface object definition,
 - about 452
- control types
 - about 903
 - custom HTML type, about SWE
 - using 904, 908
 - new HTML type, creating and using 903
- controls
 - association applet object type,
 - described 710, 711
 - docking, preventing 113
 - docking/undocking 113
 - hiding 113
 - list applet 478
- copying
 - about inheriting behavior 156
 - cascade copy, constructing with multi-value link 417
 - objects, guidelines for 970, 971
- CREATED system column, about 223
- CREATED_BY system column, about 224
- Creating the Development Environment 44
- Criteria Values and Skills, configuring 681
- CSSFrameList class, list applet Class property 474
- CSSSequence class
 - Sequence business component,
 - about 375
- currency popup applet
 - ellipsis icon, providing for 468
- currency values, using in text box 467
- custom columns
 - database, adding to 257
- Custom extension columns 658
- custom extension columns
 - configuring, setting properties 218
 - extension table and base table 219
 - in a base table, described 61
- custom extension columns in an extension table described 61
- custom extension tables, described 198
- customized repositories

- note, about using Application Upgrader to merge 967
- CX_extension table, about 260

D

- data
 - EIM and imported data 673
 - exporting 673
 - importing 673
 - MLOV upgrade log file, about 671, 672
 - upgrading bounded columns 665
- Data (Private) property
 - extension columns, about adding to 266
- Data (Private) value
 - Type property of Column object type 217
 - Type property of Table object type 266
- Data (Public) value, Type property of Column object type 217
- data columns
 - about 218
 - described 60
 - private Column object type, about 217
- data computational routines 172
- Data Function property, Chart object type 741
- data manager, about 77
- data manipulation routines 172
- data model
 - See also* Standard Database Extensibility
 - extending 251
- data object definitions, updating
 - See also* extension tables, existing extension column
 - base table extension column data, displaying 278
 - developers, making new extensions available to 277
 - EIM, about using extensions with 272
 - Enterprise Integration Manager, creating and mapping interface columns 273
 - existing extension column, deleting 270

- extension table, creating custom table 264
- extension tables and columns, populating 290
- indexes, adding custom indexes 274
- local environment, updating 275
- LONG extension column, adding to extension table 268
- one-to-many extension table, displaying data 279
- one-to-one extension table, displaying data 279
- server database, applying and activating changes 281
- server database, ddlsync.ksh, executing 284
- server database, preparing prior to applying schema 280
- server database, propagating changes to mobile user 289
- data objects
 - business objects, mapping to (diagram) 442
 - data object types 191
- Data Objects layer
 - See* tables
 - about 58
 - data object types, described 62
 - relational DBMS implemented through, about 59
 - tables and columns, described and list of 60
- Data Point Field property, Chart object type 740
- data sources
 - system data sources, creating 40
- data tables 60, 196
- data transport routines 173
- data validation routines 172
- data, blank spaces, result of defining column as Char 216
- Database Extension Designer 155
- database extension designer (Dynamic Database Extensions)
 - about and extension capabilities 253
- Database Extension Designer, about using to edit picklists 661
- database extension planning and design
 - about 256
 - custom column, planning for adding to database 257
 - DBMS restrictions 261
 - extension tables and columns, naming conventions 260
 - mobile clients, about accommodating for 261
- database extension, implementing
 - about 262
 - base table extension column data, displaying 278
 - custom extension table, creating 264
 - developers, making new extensions available to 277
 - Enterprise Integration Manager, about using extensions 272
 - Enterprise Integration Manager, creating and mapping interface columns 273
 - Enterprise Integration Manager, deleting/deactivating mappings 274
 - existing extension column, renaming 269
 - existing extension table, deleting 271
 - existing extension table, modifying 270
 - extension column, adding to extension table 266
 - extension tables and columns, populating 290
 - indexes, adding custom indexed 274
 - local environment, updating 275
 - LONG extension column, adding to extension tables 268
 - one-to-many extension table, displaying data 279
 - one-to-one extension table, displaying data 279

process steps for implementing 263
 projects, checking out and locking,
 about 263
 server database, applying and activating
 changes 281
 server database, ddlsync 284
 server database, preparing prior to
 applying scheme 280
 server database, propagate changes to
 mobile users 289
 databases
 See repositories, migrating between
 databases
 local databases, renaming and
 moving 41, 43
 mobile, upgrading 950, 951
 date
 and time, setting 135
 changed date, updating manually 136
 Date data type, Physical Type of Column
 object type 217
 Date Time data type, Physical Type of
 Column object type 217
 date time data, using in text box 468
 date values, using in text box 468
 date-time stamps, columns that provide
 for 224
 DB2, note, length of table names 362
 DBMS, extension columns, restrictions on
 adding 261
 ddlsync.ksh, executing 284
 Default property, Column object type 216
 DEFAULT value, Auto Primary
 property 415
 DELETE value, Cascade Delete
 property 399
 deleting
 custom dock objects 333
 extension tables or columns 270
 repository 933
 Denormalization Path property, specifies
 table and column names of duplicated
 column 217
 Denormalized value, Type property of
 Column object type 217
 Descendents Visualization View
 described 145
 example and described 152
 Destination (foreign key) field, about 397
 Destination Business Component property
 cascade copy feature, used to
 implement 417
 Multi Value link object of the multi-value
 link 406
 Multi-value link of the indirect multi-
 value link 409
 Destination Column property
 Join Specification object type, about 383
 Primary key column, about 384
 Destination Field property
 destination field, specified in 397
 foreign key field, specified in 407
 Link object type, about based on 397
 parent ID field, specified in 362
 Destination Link property
 Business Component object type 406
 cascade copy feature, used to
 implement 417
 Multi Value Link object definition,
 referenced in 403
 Multi Value link object of the multi-value
 link 406
 Multi-value link of the indirect multi-
 value link 409
 Destination Table property, about 232
 detail applet
 association applets, (Opportunity Contact
 list applet) object type 709
 records displayed in the association
 applet 710
 Detail applet, master-detail view object
 definition 492
 detail business component
 about 209
 business object, about 425
 indirect multi-value link, about 410

- Link object definition, about 397
- master-detail business components, about 361
- master-detail view object definition 493
- multi-value link object definition, about 406
- records displayed in the association applet 710
- Search Specification property, about 362
- Detail tab
 - described and example 103
- detail View bar
 - about and rules for context views for a selected view 804, 805
 - SWE template configuration, about and code syntax 806, 807
- Details Visualization View
 - business component, generating a view 148
 - business object version, example and described 147
 - described 145
 - shortcut menu, displaying 148
- developers
 - Data Object layer columns 61
 - database extensions, making available to 277
 - mobile users, setting up as 45, 47
- development environment
 - creating 44
 - mobile users, setting up developers as 45
 - repository naming convention guidelines 43
- Development Tools Options dialog box, check in/check out options (data sources), described 1005, 1007
- directories, location and contents 913
- directory structure, verifying 38
- Display Format property
 - Control object type, list of properties 467
 - explicit format mask, about specifying 484
 - List Column object type 477
- Display Name property
 - List Column object type 476
 - Tree Node object type 761
- Display Value
 - field described 685
 - language-independent code, about columns enabled for MLOV 676
 - queries, about using to run 676
- displays, multilingual, special cases 662
- Dock Object Visibility Rule object type 316
- Dock Object Wizard
 - flow diagram 322
 - running in Tools 320
- dock objects
 - about 312
 - deleting, about 333
 - Dock Object Table object, about and example 314
 - dock object visibility rules, about and example 316
 - Dock Object Wizard, flow diagram 322
 - Dock Object Wizard, running in Tools 320
 - dock objects, dock object tables, and dock object visibility rules, verifying 332
 - existing dock object, adding a new dock table to 328
 - Limited dock objects, visibility rules 313
 - new dock object, creating 325
 - type of 312
- docking windows
 - preventing when moved 113
 - procedure 113
- DOS command prompt
 - MLOV Upgrade Utility, starting from 668
- drilldown behavior
 - Drilldown object type, about and example 494
 - dynamic drilldown, about and configuration (diagram) 496

static drilldown, about and configuration (diagram) 495
drilldown fields, text field with 473
drilldown, dynamic 675
driving table, about 314
drop-down arrow buttons, about attaching to text box 467
DTYPE_BOOL data type, described 370
DTYPE_CURRENCY data type, described 370
DTYPE_DATE data type, described 370
Display Format property of Control object type 468
DTYPE_DATETIME data type, described 371
DTYPE_ID data type, described 371
DTYPE_ID, note, multi-value group applet rules 700
DTYPE_INTEGER data type, described 371
DTYPE_NOTE data type, described 372
DTYPE_NUMBER data type, described 372
DTYPE_PHONE data type, described 372
DTYPE_TEXT data type, described 372
DTYPE_TIME data type, described 373
Dynamic Database Extensions, about and extension capabilities 253
dynamic drilldown, about and configuration details (diagram) 496

E

eBriefings, using to manage Web content, about 186
Edit Definition option, Visualization View shortcut menu 148
Edit Layout option, Visualization View shortcut menu 148
Edit List mode for applets described 793
Edit List mode, showing list applets 834, 836
Edit mode
applets, described 793
form applet, showing 832

EIM
extensions, about using with 272
extensions, creating and mapping interface extension columns 273
extensions, deleting/deactivating mappings 274
EIM attribute mappings, note, modifying 228
EIM base tables, note, modifying user keys 227, 232
EIM Interface Table Column object type, about 231
EIM Interface Table object type, about 230
EIM Interface Table project 256
EIM interface tables
about 228
EIM object types architecture (diagram) 229
loaded data labeled as NULL, fixing 238
mappings, restrictions on adding and modifying, table 237
object types used in configuration described 229
EIM prefix. *See* EIM interface tables
EIM Table Mapping list
note, new extension table, adding to 269
EIM Table Mapping object type, about 232
EIM table mapping objects
EIM interface column, attribute mapping objects for each column generated 309
EIM interface table 303
EIM interface table columns 303
EIM processing, generic EIM interface table columns for 304
foreign key map, Foreign Key Mapping Column for each 311
foreign key processing, EIM interface table columns for 306
foreign keys, EIM interface table columns for 307
table, EIM interface table columns for processing mapping for 305

- target table, EIM table mapping object
 - based on selected table 309
- target table, foreign key mapping for each
 - foreign key column on 310
- target table, attribute for each EIM
 - interface column on the 308
- EIM Table Mapping Wizard
 - See also* EIM table mapping objects
 - accessing and about 299
 - adding 300
 - flow diagram 300
- Enterprise Integration Manager, about using
 - to import and export data 673
- Enterprise Integration Manager. *See* EIM interface tables
- environment
 - local environment, updating 275
- ERP Interface Table project 256
- error messages
 - forms, displaying within form 911, 912
- errors
 - LOV type messages 671, 672
- eSmartScript, configuring for dynamic data
 - capture 186
- existing extension column, modifying 269
- Expand All option
 - Visualization View shortcut menu 148
- explorer applet, about and example 460
- explorer view. *See* tree applets
- export process 673
- exporting
 - export file, contents of 918
 - individual object definitions 919
 - projects 922
 - repository 935, 939
 - repository, about adding a repository
 - object definitions to an export list 923
 - repository, using repimexp.exe 939
- extension column names
 - about and examples 261
- extension columns
 - configuring 269
 - configuring, setting properties 218
 - described 61
 - extension columns and foreign keys 221
 - note, using database reserved words 261
 - populating 290
 - standard extension columns, about and
 - table of 220
 - value and types of extension
 - columns 218
- extension table
 - custom extension table, creating 264
- extension tables
 - base table, two different contexts used
 - in 194
 - business components, about 346
 - configuring 269
 - custom extension columns,
 - described 61
 - described and relationship with other
 - tables (diagram) 196
 - displaying data in 279
- Enterprise Integration Manager, about
 - using with 272
- Enterprise Integration Manager, creating
 - and mapping interface extension
 - columns 273
- Enterprise Integration Manager, deleting/
 - deactivating mappings 274
- existing extension column, deleting 270
- existing extension column,
 - modifying 269
- existing extension column,
 - renaming 269
- existing extension table, deleting 271
- existing extension table, modifying 270
- extension column, adding to 266
- implied joins, about, example, and
 - definitions 201
- LONG extension column, adding to 268
- naming conventions 260
- one-to-many extension tables, about and
 - example 203

- one-to-one extension tables, diagram and object definitions 200
 - populating 290
 - Siebel 2000, created in 260
 - standard and custom extension tables, distinctions between 197
 - table object definition, described 60
 - types of 199
 - Extension value
 - Type property of Column object type 217
 - Type property of Index object type 226
 - external business components
 - detail business component, object definitions 361
 - link object definition 361
 - master business components, object definitions 360
 - one-to-many extension table details (diagram) 360
- F**
- field data types
 - about and example 370
 - table of 370
 - field object definition, described 66
 - Field property
 - Control object type 467
 - Control or List Column object, specifies field 364
 - form applet control property 535
 - List Column object type 476
 - Field Read Only Field user property
 - about, properties, and example 353
 - restricting field procedure 353
 - fields
 - about, form and list applet examples 362
 - calculated fields, about 367
 - calculated fields, data types, specifying 368
 - search specification, in a 349, 487
 - sequence fields, about and object definitions 373
 - sequence fields, adding sequence field to business component, procedure 376
 - system fields, about and fields and columns, table of 366
 - file attachment applets
 - about, capabilities, and example 765
 - attachment applets, about using for attachment support and synchronization 768
 - attachment business component, requirements and fields 769
 - attachment tables, about file columns (table) 771
 - document, adding to attachment list 766
 - file attachment applets, about and example 461
 - Fill color, Chart Element object type 750
 - filtering a pick applet, about and example 632
 - find
 - Applet Web Template, object properties 828, 829
 - search and find applet tags, list of tags and example 875, 877
 - Search/Find results list applet, list of tags and example 877, 878
 - Find combo box
 - association applets, in 711
 - Find Field object type 341
 - Find in Repository window, user interface elements 141
 - Find object type 341
 - Find objects, about configuring 512
 - Find View object type 341
 - Flat tab, described and example 104
 - Font, Chart Element object type 750
 - Foreign key field
 - indirect multi-value link (in the detail business component), about 410

indirect multi-value link (in the joined business component), about 410
multi-value link, about 407
Foreign Key Mapping Column object definition, about 236
Foreign Key Mapping object definition, about 234
foreign keys
 caution, using extension columns to track 268
 column, about 228
 extension column, about configuring 221
 Join object definition, about 384
form applet (Account Form applet) object type, about 714
form applets
 about and example 454
 architecture (diagram) 463
 button control properties 469
 check box controls, about and example 470
 combo box controls, about and example 470
 control properties, list of 535
 controls, list of 465
 Edit mode view, showing form applet in 832
 label controls, about and example 472
 object types and roles in implementing 463
 persistently editable forms, about using and screen example 832
 text boxes, characteristics 466
 text boxes, properties 467
Formatting templates, template type 794
forms, displaying within error messages 911, 912

G

GIF formats, about using 893
Go button, in association applets 711

Graphic value, Chart Element object type 750
grid layout
 about 547
 applet Web templates 900
 converting applets to 548
 editing 555

H

high interactivity
 about 80
HTML files, files that define layout and Siebel Web Engine, about 73
HTML frames
 about and screen example 817, 819
 about using in View templates 825, 826
 support for multiple views on a page 822, 825
 swe template configuration 819, 820
 using container page template 820, 821
 using in View templates 826, 828
HTML Hierarchy bitmap, about and properties 526
HTML type
 custom HTML type, about SWE using 904, 908
 definitions, changing the order of precedence 908
 new HTML type, creating and using 903
 templates used to create custom HTML types 794
HTML Type property of List Column object type, about and example 540
HTML Type property, List Column object type, about 477

I

Icon Map object type, about and rendering fields as 524
Icon Map, using to render control/list column field values to icons 894, 896
Id field

implied join definition, described 202
 ROW_ID column, about
 representing 223
 IFMGR: xxx value, Type property of Column
 object type 218
 images
 Bitmap object, about configuring and
 example 892
 control/list column field values,
 rendering as icons 894
 controls/list columns, using images as
 links in 897
 GIF and JPG formats, about using 893
 image caching file manager,
 parameters 898
 previewing 114
 implied joins, about, example, and
 definitions 201
 importing
 object definitions and properties,
 adjusting conflict resolution 927
 object definitions from an archive file,
 process of 923
 object definitions, loading from an
 archive file into the Preview
 window 924
 repository 935, 939
 repository, using repimexp.exe 939
 target repository, conditions before
 importing 923
 importlog.txt, about 931
 Index Column object type
 about and properties 226
 index object definition
 about and key properties, list of 225
 index column object type, about and
 properties 226
 note, length of index names 227
 index object definition, described 62
 indexes, adding custom indexes 274
 indirect multi-value links
 construction of and object
 definitions 408
 joins, using to populate MVG 412
 parent components, indirectly related to
 business component 411
 indirect multi-value links, note, using in
 place of multi-value link 694
 Informix, note, length of table names 362
 inheritance behavior, about copying and
 inheriting 156
 installation
 development environment, setting
 up 44
 directory structure, verifying, example
 and described 38
 installation successful, verifying 37
 local databases, renaming and
 moving 41
 Microsoft Data Access Components,
 installing 36
 ODBC data sources, created during
 install 41
 pre-install tasks 31
 repository naming conventions, about
 and guidelines 43
 workstation, installing on 32
 Integration Object objects
 wizard, about using to create 446
 Inter Child Column property
 about 210
 destination field properties, about
 specifying if field non-blank 397
 Inter Parent Column property
 destination field properties, about
 specifying if field non-blank 397
 Inter Parent Column property, about 209
 Inter Table property, about specifying if field
 non-blank 397
 interface extension columns
 creating and mapping 273
 deleting/deactivating mappings 274
 interface standard Siebel 159
 interface table columns, note, about
 multiple columns to single
 column 237

- Interface Table User Key Usage object type 232
 - interface tables
 - See also* EIM interface tables
 - described 60
 - special columns relating to EIM processing 218
 - intersection business components
 - about and example 356
 - intersection data columns 212
 - intersection table, described 60
 - intersection tables
 - Advanced Database Extensibility, between existing tables 292
 - architecture of (diagram) 207
 - configuring, about and architecture 207
 - described and many-to-many relationship 204
 - example with two links 211
 - extending, about 213
 - intersection business components, and 356
 - intersection data columns, described and examples 212
 - link, about based on 397
 - links used for many-to-many relationship, property settings and diagram 211
 - object definitions 209
- J**
- J2EE environment, about integrating Siebel with 85
 - java, about integrating Siebel with 85, 86
 - Java/XML Framework, about using to receive XML requests from Siebel 86
 - JavaBeans, about integrating with Siebel 86
 - JavaScript object architecture, described 81, 83
 - Join business component, about 409
 - Join Constraints, about 384
 - Join object definition
 - indirect multi-value link, about 410
 - join object definition, about 383
 - join object definition, described 67
 - Join property, Field object type 383
 - join specification object definition
 - described 67
 - indirect multi-value link 410
 - join object definition 383
 - join specification, pick list object type 619
 - join, pick list object type 619
 - joined field
 - indirect multi-value link, about 410
 - join object definition, about 383
 - joined tables
 - business components, about 345
 - join object definition, about 384
 - joins
 - about and example 378
 - master-detail relationship, described and diagram 378
 - object definition relationships (diagram) and object definitions 382
 - pre-default value, using for a join field 385
 - rows resulting from joins (diagram) 380
 - JPG formats, about using 893
- K**
- Key field
 - indirect multi-value link, about 410
 - master-detail business component, about 360
 - multi-value link, about 406
 - keyboard accelerators, configuring 594
 - ksh, executing 284
- L**
- label controls, about and example 472
 - Label Field property
 - Tree Node object type 762
 - Language column, described 684
 - language-independent code

- applet, creating for Workflow Policy
 - Column 677
- applet, creating for Workflow Policy
 - Program Argument 679
- avoiding in queries 676
- deleted LOV records 686
- display of 674
- and Display Value 676
- display value translation 664
- exporting data 673
- List of Values use 684
- picklist, creating for a Workflow Policy
 - Column 677
- picklist, creating for Workflow Policy
 - Program argument 679
- and querying 676
- and S_LST_OF_VAL 656
- sorting and searching 674
- target column display 674
- LAST_UPD system column, about 224
- LAST_UPD_BY system column, about 224
- Legend value, Chart Element object
 - type 750
- LIKE operator
 - search specification, in a 349, 487
- Limited dock objects
 - visibility rules 313
- line charts
 - 2dLine, about and screen example 730
 - 2dSpline, about and screen example 731
 - 3dLine, about and screen example 730
 - 3dSpline, about and screen example 732
 - Combo line, about and screen
 - example 733
- link
 - master-detail view object type,
 - about 493
- Link destination fields, note, about
 - automatically initialized 394
- Link object definition
 - links object type, about 396
 - master-detail business components,
 - about 361
 - multi-value link, about 406
- Link object type
 - business objects, about 425
 - indirect multi-value link, about 410
 - intersection tables 209
- link object type
 - described 66
- Link property, Business Object component,
 - about 424
- links
 - about, master-detail relationship and
 - used in multi-value group
 - applets 393
 - architecture (diagram) 398
 - Cascade Delete property, about and
 - values 399
 - intersection table, based on 695
 - many-to-many relationship, about using
 - in 399
 - master-detail views, about using 398
 - merging records, about using a link
 - when 399
 - multi-value group applet, about and
 - properties 695
 - multi-value group applet, about using
 - in 399
 - multi-value group object type,
 - about 691
 - object definitions, list of 396
 - relationship between object definitions
 - (diagram) 396
- List Applet Wizard, using to create list
 - applet 536
- list applets
 - about and examples of cell behavior 472
 - about and screen example 833
 - applet style described 455
 - architecture (diagram) 474
 - configuring the list, about 476
 - control properties, list of 535
 - controls, about configuring and
 - descriptions of controls 478

- Edit List mode, showing list applets
 - in 834
- explicit format mask, about
 - specifying 484
- list applet definition, steps in creating (diagram) 532
- List Applet Wizard, using to create 536, 538
- List Column, configuring 476
- lists and list columns, about 540
- multi-record select applets, about and screen example 845
- multi-record select applets, enabling 845
- multi-record select applets, rendering
 - checkboxes to select multiple rows 846
- multi-row edits, about and limitations 844
- multi-row edits, enabling 844
- record, selecting as active record 840
- testing configuration without user-specified settings 539
- totals of list column values, two methods of showing 846 to ??
- type properties, list of 485
- list column
 - field values, rendering as icons 894, 896
 - images, using as links in 897
- list column object definition
 - applet object type, about 475
 - described 73
 - user interface type, about 452
- List Column object type, about and example 540
- List Column, configuring 476
- list columns
 - about in list applet 472
 - association applet object type 710
 - control behavior, enhancing 540
- list object definition
 - applet object definition 475
 - described 73
- List property, list applet control
 - property 535
- list, user interface object 452
- list-form view
 - about 488
 - applets, property setting in Account List View 490
 - list-form, about and example 489
- LMU. *See* Locale Management Utility
- local databases, renaming and moving 41, 43
- local environment, updating 275
- Locale Management Utility
 - See also* localization
 - about using 601
 - modified objects, identifying 608, 609
 - note, about modified records for project marked as Redo 608
 - note, about not marking changed records in Preview mode 606
 - strings and other locale-specific attributes, exporting 605
 - strings and other locale-specific attributes, importing 605, 607
 - untranslated text strings, finding 602, 603
- locale object, about finding untranslated text strings 602
- locale-specific attributes
 - exporting 605
 - importing 605, 607
- localization
 - See also* Locale Management Utility
- locking projections
 - when extending the database through Siebel Tools 256
- locking projects
 - local projects 1011, 1012
 - note, displaying projects that need to be locked 924
 - server projects 1012
- locking when extending the database through Siebel Tools 256

log file 671, 672
log file, Summary window messages, list of 931
logical operators
 search specification, in a 349, 486
logical schema, updating. *See* data object definitions, updating
logical UI object types, described 72, 73
login, specifying explicit login ID 183
logon stamps, columns that provide for 224
Long data type, Physical Type of Column object type 216
LONG extension column, creating 268
LOV table
 See language-independent code administration columns 684
 constraints on entries 686
 display value 685
 EIM validation 685
 enabling for multilingual LOV (MLOV) 658
 entries, constraints on 686
 Language column 684
 multilingual integration considerations 673
 records, adding 685
 records, deactivating 685
 records, deleting 685
 Translatable column 684
LOV Type
 See also LOV table
 enabling multilingual support 663
 enabling target columns for MLOV 663
 error messages about 671, 672
 fixing from log file 671
 multilingual designation 684
 Multilingual property 663
 querying 660
 and translation 659
LOV_Upgrade.log 671, 672
LOVs

domains not in S_LST_OF_VAL table 672
inconsistently bounded columns 661, 671
multilingual, about enabling 657

M

M:M link, about specifying source field 397
many-to-many relationship
 links, about using in 399
many-to-many relationship. *See* intersection tables
many-to-one extensions
 Advanced Database Extensibility, about adding to existing tables 292
Mapped column, about 384
mappings
 adding and modifying, restrictions on 237
 business components to business objects 441
 deactivating 237
 screens to screen menu items 578
master applet (Opportunity form applet) object type, about 709
Master applet, master-detail view object definition 492
master business component
 about 209
 business object details, about 425
 indirect multi-value link, about 410
 link object type, about 396
 master-detail business components, about 360
 master-detail view object definition 493
 multi-value link, about 405
master-detail view
 about 488
 about and example 490
 Contact Form applet and Contact Account applet, about 491
 links, about using in 398

object definitions, list of 492
 MDAC (Microsoft Data Access Components), installing 36
 menus
 about and figures 513
 applet-level menus, about invoking and screen example 865
 menu items and toolbars, activating and suppressing 520
 method targeting, invoking 521
 SWE templates, menu configuration 868
 toolbar and menu-related object types 515
 Method Invoked property
 Control object type 469
 form applet control property 535
 Microsoft Data Access Components, installing 36
 Microsoft Visual SourceSafe
 examples 964
 repository management, about using in 957
 Microsoft Web server software, interfacing with the Siebel Web Engine 78
 Microsoft Windows Control Panel, note, formatting field data types 369
 MLOV
 Assignment Criteria and Skills, configuring 681
 configuration and coding guidelines 675
 configuration for data read by server programs 658
 drilldown, dynamic 675
 enabling, about 657
 querying 676
 search specifications 675
 and Siebel Anywhere 683
 and Siebel Assignment Manager 681
 and Siebel Field Service 683
 toggle applets, coding guidelines 675
 and Translate property 660
 upgrade utility, about 665
 VB functionality, about 675
 visibility rules, checking 659
 Workflow Manager, configuring for 676
 MLOV table
 administering MLOV entries 684
 administration columns 684
 columns, identifying which to enable 658, 659
 configuration compatibility 674
 displaying columns, warning about 662
 EIM validation 685
 multilingual designation 684
 multilingual integration considerations 673
 Multilingual property 663
 records, adding 685
 records, deactivating 685
 records, deleting 685
 target columns 663, 664
 translating display values 664
 visibility rule changes 658
 MLOV Upgrade Utility, starting from DOS Prompt 668
 mobile databases, upgrading 950, 951
 mobile users
 See dock objects
 developers, setting up as 45, 47
 server database, propagating changes to 289
 Multi Value Field
 object type, viewing 435
 multi value link
 object type, about 406
 multi value link, described 67
 multi value list
 text box, about popping up list 467
 Multilingual column, described 684
 multilingual displays, special cases 662
 multilingual lists of values. *See individual MLOV entries*
 Multilingual property, LOV type 663
 multi-part tags, about and example 797, 798
 multi-value fields

- indirect multi-value link, about 409
- master business component, about 403
- master-detail business component,
 - about 361
- multi valued property, with setting of true 369
- multi-value group object type 691
- multi-value link, about 406
- multi-value group applet
 - object type properties 699
 - pick applet, about configuring when invoked from MVG applet 635
- multi-value group applet (Industry Mfg. applet) object type, about 714
- multi-value group applets
 - See also* association applets
 - about and example 456, 688
 - implementing, links used 394
 - links, about using in 399
 - multi-value group business component,
 - about storing data and identifying primary record 695
 - MVG architecture (diagram) 689
 - object definitions (diagram) 690
 - object type, about 691
 - object types, list of 691
- multi-value group business component (Industry) object type, about 715
- multi-value group business component, multi-value group object type 692
- multi-value groups
 - defined 400
- multi-value links
 - See also* indirect multi-value links;
 - Primary ID field
 - about and examples 400
 - architecture, described and diagrams 403
 - cascade copy, about constructing 417
 - indirect multi-value link, about 409
 - master-detail business component 361
 - multi-value group applet, about and properties 693

- multi-value group applet, using to modify Category 694
- multi-value group type 691
- note, using indirect multi-value link 694
- object definition, list of 405
- MVG Applet property
 - Control object type, about 469
 - List Column object type, about 477
- MVG Applet Wizard
 - configuring a MVG wizard 696
 - using to configure applets 623, 626

N

- Name field, about 361
- Name property
 - about 216
 - Business Component object type 432
 - Business Object object type 440
 - Control object type, about 465
 - form applet control property 535
 - Index object type 225
 - List Column object type, about 476
 - Single Value Field object type 434
 - Single Value Field object type, about 534
 - User Key Attribute object type 236
 - value, about changing 442
- navigation
 - first level, primary tab bar 803
 - fourth level 809, 813
 - levels of navigation, described 802
 - levels, screen examples 802
 - second and third levels 804
- nesting, Siebel tags, about and example 800
- New button, in association applets 710
- New mode for applets, described 793
- New Object wizards 93
- Newtable project 256
- No Copy property
 - cascade copy feature, used to implement 418
 - Single Value Field object type 434
- No Delete property

Business Component object type 432
Single Value Field object type 534
No Insert property, Business Component
object type 432
No Insert property, Single Value Field object
type 534
No Update properties, Business Component
object type 432
No Update property, Single Value Field
object type 534
NONE value
Auto Primary property value, about 415
Cascade Delete property, about 400
NOT operator, search specification, in
a 349, 486
null records, appear in record-set 348
Number data type, Physical Type of Column
object type 216
number data, displaying in text box 467

O

object comparison
See *individual repositories item*
about 975
archive files, comparing two 985
differences between checked-out projects,
viewing 981
differences, viewing between two
objects 977
repository, selecting 985
.sif file, using for comparison with
object 984
two objects of the same type, ways to
compare 983
Object Comparison window,
invoking 1008, 1009
Object Compiler dialog box
accessing 130
caution, about compiling or modifying
.srf file 130
objects, compiling single or groups
of 131
object definitions
See also importing; exporting; Data
Objects Layer; objects; queries
about and examples 49
caution about changing 664
changed date, updating manually 136
Check In dialog box, user interface
elements, described 1003, 1004
check mark and pencil icons, meaning
of 135
created and last update, determining by
whom and when 135
defined 51
exporting individual object
definitions 919, 920
in Data Objects layer 63
intersection tables 209
new object definition, creating 121
new object, creating from copy of existing
object 120
note, property column, moving to 119
Object Explorer window Detail tab,
displaying all object definitions 103
Object List Editor, modifying 118
Object List Editor, viewing through 114
parent-child relationships, displaying
object definition and its parent\child
definitions 153
project structure guidelines 1012
projects, assigning to 999
projects, checking in new or modified
object definitions 1001
projects, moving between 1000
Properties window, using to modify 119
Properties window, viewing
through 115
reassigning object definitions 999
report, restricting to a single parent object
definition 143
undoing record or modification 123
Validate window user interface
elements 125
validating procedure 123
validation options, changing 126

- value types, list of and examples 117
- visualization diagram, selecting object definition properties 148
- Object Differences windowpane
 - about 928
 - object definitions, using to view hierarchy of object definitions at check in 1009
- Object Explorer window
 - docking, preventing 113
 - docking/undocking 113
 - hiding 113
 - making visible 100
 - object definitions, about displaying in the Object List Editor window 107
 - tabs in 101
 - visible object types, configuring 105, 106
 - visible, making 99
 - window example 98
 - windows, available to make visible 108
- Object Explorer, viewing Business Component child object types 435
- object layers
 - architectural layers diagram and layers described 57
 - object types in Siebel apps and relationships between (diagram) 75
- Object List Editor window
 - See also* queries
 - about 107
 - note, about docking or floating, default setting 113
 - object definitions, using to modify 118
 - object definitions, viewing through 114, 116
- Object List Editor, using to define view 564
- object manager, about 77
- object relationships
 - business components\business objects, mapping to underlying tables, display of 146, 148
 - Visualization Views, list of 145
- Visualization Views, two methods of invoking 145
- object types
 - about 49
 - defined and parent-child relationships 52
 - filtering 106
 - major in Siebel apps and relationships between (diagram) 75
 - Object Explored window Types tab, accessing from 102
 - Object Explorer window Types tab, accessing from 101
 - parent and child without relationship, accessing 104
 - reports available for each object type 144
 - visible in object explorer, configuring 105
- objects
 - adding, modifying, copying 117
 - comparison and synchronization, about 156
 - descendents or copies, viewing 973, 974
 - new object, creating from copy of existing object 120
- objects, identifying modified objects 608, 609
- ODBC data sources
 - install, created during 41
- OleDb Rowset wizard
 - about 445
- OleDb specification, about 445
- one-to-many extension table
 - data, displaying 279
 - Search Specification property, about 362
- one-to-many extension tables
 - about and example 203
 - rows, about 199
- one-to-many relationships
 - See also* business objects; links
- one-to-one extension tables
 - data, displaying 279

- diagram and object definitions 200
 - implied joins, about, example, and definitions 201
 - rows, relationship with rows in base table 199
- one-to-one extensions
 - Advance Database Extensibility table 292
- one-to-one relationships, and joins 381
- OR operator, search specification, in
 - a 349, 486
- originating applets
 - multi-value group applet properties 692
 - multi-value group object type, about 691
 - pick applet, about, detail diagram, and property settings 620
 - pick list object type, about 618
 - Static pick list, property settings 643
 - static picklist object type 641
- originating business components
 - about and detail diagram 627
 - multi-value group applet, about and properties 692
 - multi-value group object type, about 691
 - pick list object type, about 618
 - property settings 629
 - static picklist object type, about 642
 - static picklist, about and property settings 643
- originating control
 - pick list object type 618
 - static picklist object type 642
- originating field
 - pick list object type 618
 - static picklist object type 642
- originating list column
 - pick list object type 618
 - static picklist object type 642

P

- Page Container template, template type 794
- page tab object definition
 - described 72
 - user interface type, about 451
- Page Tab object type 341
- page tabs, creating in application 577, 578
- PAR_ROW_ID column
 - implied join definition 203
 - system column, about 224
- PAR_ROW_ID column, about 362
- Parent Business Component property
 - Link object type 396
- Parent ID field, about 362
- Parent Read Only Field
 - about, properties, and example 354
- parent-child relationships
 - object definitions, viewing relationship between 145
- parent-child relationships, defined 53
- party 240
- password encryption
 - checking out projects, problem using 996
- patch file
 - See also *individual repositories item*
 - about, differences from archive file, and process diagram 951
 - creating 953
 - note, about creating cumulative patch files 954
- pencil icon
 - See also date, and time setting; object definitions
 - meaning of 135
- performance
 - note, search specifications, impact on 350, 488
 - strategies to optimize application performance 169

performance considerations, chart
 applets 751

period charts, about 746

phone data, using in text box 468

Physical Type property
 about 216
 note, about modifying property for
 columns 270

Physical User Interface layer
 described 74

pick applet object type
See also static picklists
 about 618
 about, detail diagram, and property
 settings 622

Pick Applet property
 Control object type, about 469
 List Column object type, about 477

Pick Applet Wizard, using to configure
 applets 623

pick applets
 about and example 456, 613
 architecture (diagram) 617
 data flow in pick applet 615
 filtering, about and example 631, 632
 object definitions, list of 618
 using, example 615

pick applets, about, detail diagram, and
 property settings 634

pick business component
 pick applets, about, detail diagram, and
 property settings 633
 pick list object type, about 618

pick list
 object type, about 618
 Pick List wizard, using to create a
 picklist 645

pick list object definition
 filtering a pick applet, about and
 example 631
 pick list, about, detail diagram, and
 property setting 634
 static picklist, about and properties 644

pick list object, pick list object type 642

Pick List wizard, using to create a pick
 list 645, 647

pick maps
 note, about using multiple pick
 maps 644
 pick applet object type 619
 static picklist object type 642

PickList Generic business component
 pick applet object type 647, 648
 static picklist object type 642

Picklist Types property
 Chart object type 742

picklists
See also chart applets, configuring
 bounded 660
 display value 685
 language-independent code,
 creating 677, 679
 list applet type property 485
 multilingual LOVs, about enabling 657
 pick list object definition, about, detail
 diagram, and property settings 634,
 635
 repository, recompiling after
 changes 673
 text box, about popping up list 467
 unbounded 660
 unbounded, about type in new
 values 630
 upgrading bounded columns 665

pie charts
 3dPie, about and screen example 734

Plot value, Chart Element object type 750

Policy Conditions, enabling 677

pop-up
 list applet type property 485
 static picklist, visibility rules 650

popup applets
 applet, launching from within 773
 CSSSWEFramePopup, about using 772
 multi-value group applet, behavior
 of 849

- popup view, launching from applet 775
 - Wizard style popup. about and example 774
- Popup Edit property
 - Control or List object type 372
- Popup Visibility Type property, about using to limit picklist visibility 650
- Position property
 - Tree Node object type 762
- postal code formatting options, about 468
- Pre Default Value property
 - Field object type 362
- Precision property, about 217
- pre-defined queries
 - configuration and SWE tags 879
- pre-install tasks 31
- Primary ID field
 - about using and example 413
 - multi-value link object type 407
 - No Match Property, using with 416
 - object definitions 414
- Primary Id Field property
 - Multi Value link object of the multi-value link 406
 - Multi-value link of the indirect multi-value link 409
- Primary key (destination) column, about 384
- Primary Key property, about 217
- Primary Key value, Type property of Index object type 226
- primary tab bar
 - described 803
 - swe:screenbar, about and code snippet 803, 804
 - tools configuration 803
- private data columns, about 217
- Private dock object, about 312
- production environment, repository naming convention guidelines 43
- Project List report, described 144
- Project property, note, about exporting/importing from an archive file 918
- projects
 - See also* compiling projects; importing; exporting
 - caution, about compiling or modifying default ,srf file 130
 - caution, about incremental compilations 130
 - Check In dialog box, user interface elements described 1003
 - check in guidelines, list of 1004
 - check in/check out options (data sources) 1005
 - Check Out dialog box, user interface elements described 996
 - checked out, undoing 1009
 - checked out projects, viewing differences 981, 982
 - checking in projects 1001
 - checking out copy 995
 - defined and about (screen example) 987
 - deleting, about 998
 - differences at check in, viewing details of changes 1008
 - differences at check in, viewing hierarchy definitions changes 1009
 - EIM Interface Table 256
 - ERP Interface Table 256
 - exporting 922
 - full get, described 994
 - initial get, described 994
 - locking local projects directly 1011
 - locking server projects directly 1012
 - new projects, creating and when to add new projects 998
 - Newtable 256
 - note, displaying projects that need to be locked 924
 - object definitions, assigning to 999
 - object definitions, moving between projects 1000
 - object definitions, viewing by projects 992
 - project structure guidelines 1012

- renaming 999
- repository, doing an initial get of all projects 994
- Scopus Migration 256
- suffix names, meaning of 988
- properties
 - defined 51
 - docking, preventing 113
 - docking/undocking 113
 - hiding 113
 - settings, using Properties window to display 108
- Properties window
 - described 108
 - object definitions, using to modify 119
 - object definitions, viewing through 115, 116
- PushButton control, about 73

Q

- QBE. *See* queries
- queries
 - compound queries, about creating and table of 139
 - object definitions, about using to list 137
 - Object List Editor query, creating and executing 137
 - Object List Editor window, restoring to pre-query state 138
 - pre-defined queries, configuration and SWE tags 879
 - simple operators, table of 138
- queries, about running against MLOVs 676
- Query List Business Component property, about 440
- Query mode for applets, described 793
- query-by-example. *See* queries

R

- RDBMS, implemented through Data Objects layer 59
- Read Only property
 - Control object type 468
 - form applet control property 535
 - Single Value Field object type 434
- read-only behavior
 - about and object type user properties 350
 - user properties, warnings on 351
- record, selecting active record as 840, 843
- records
 - changed date, updating manually 136
 - check mark and pencil icons, meaning of 135
 - created and last update, determining by whom and when 135
 - merging, about using a link when 399
- recursive trees, about and properties 764, 765
- Redo, about Locale Management Utility marking projects 608
- relational DBMS. *See* RDBMS
- Relationships Visualization View
 - business component version, example and described 149
 - described 145
 - Tables version, example and described 151
 - Tables version, navigating within 151
 - View Details diagram, displaying for selected business component 150
 - View Relationships, navigating within business components version of 150
- remote clients, configuring multilingual 683
- reports
 - reports available for each object type 144
 - single parent object definition, restricting report to 143

Tables report for the S_ORG_EXT table,
 getting 143

repositories

- backing up and restoring, about 934
- deleting 933
- export, using repimexp.exe 939
- file dump, using repimexp.exe 939
- import, using repimexp.exe 939
- importing or exporting 935
- note, about compiling projects after
 changing repository 935
- note, about merging customized
 repository using Application
 Upgrader 967
- note, exporting by adding Repository
 object definition to an export list 923
- renaming 933
- reverting to previous version,
 example 964
- target repository, preparing for
 importing 923

repositories, integrating with external
 source code control

- about 957
- example, check in 964
- example, reverting to previous
 version 964
- source control interface, enabling 958
- srcctrl.bat file, configuring 959

repositories, migrating between databases

- about 942
- mobile databases. upgrading 950
- procedure 945
- processing steps 943
- target database, preparing for new
 repository 944

repositories, upgrading

- See also* upgrading inheritance
- Siebel Application Upgrader, about
 using 966

repository

- See also* reports
- about and Contact repository object
 example 54
- contents of 56
- current repository, getting information on
 and about 990
- current repository, opening and verifying
 you have the correct repository 989
- initial get of all projects, doing 994, 995
- naming conventions, about and
 guidelines 43
- object definitions, displaying and
 finding 141

Repository Dock Objects report,
 described 144

repository validator, using to detect errors in
 configuring .COM applications 170

reprimexp.exe

- parameters (table) 940, 942

reprimexp.exe, about using and parameters
 (tables) 939

Required attribute, note, about ignoring
 attribute 691

Required property

- Single Value Field object type 434

results.slf, about 605

results.txt, about 605

reverting to previous version of
 repository 964

ROW_ID Column, about and format of 223

rows, example resulting from joins 380

Runtime property, Control object type 468

S

S prefix table names, about 194

S_LST_OF_VAL

- and language-independent code 656
- LOV domains not in table 672

S_PARTY. *See* access control

Scale property, about 217

scatter chart, 2dScatter chart 735

schemas, migrating between databases

- about 942, 943
- mobile databases, upgrading 950, 951

- processing steps 943
 - target database, preparing for new repository 944, 945
 - target database, procedure 945, 950
- Scopus Migration project 256
- screen menu item object definition
 - about 451
 - described 72
- screen object definition
 - described 72
- screen user interface object type 451
- screen view
 - defining 574, 576
 - object definition, about 451
 - object definition, described 73
- screens
 - about, example, and object definition 502
 - application object definition, about and contents of 508
 - defining procedure 573, 574
 - defining, about 572, 573
 - screen menu items, associating to 578
 - user interface element, described and example 71
- scripts, Siebel VB or Siebel eScript, application object definition 508
- search
 - Applet Web Template, object properties 828
 - repository, searching for object definitions 141
 - search and find applet tags, list of tags and example 875
 - Search/Find results list applet, list of tags and example 877
- Search objects, about configuring 512
- Search Specification property
 - about 362
 - Business Component object type 432
 - sample search expressions, and syntax 348, 486 to 488
 - Single Value Field object type 534
- second By picklist
 - about 743
 - configuring, properties and examples 747
- security considerations
 - authentication 184
 - explicit login ID, specifying 183
- Selected Bitmap Index property
 - Tree Node object type 762
- SELECTED value, Auto Primary property, about 415
- Sequence Field, child object definition
 - business component user property 375
- sequence fields
 - about and sequence field configuration details (diagram) 373
 - object definitions 375
 - sequence field, adding to a business component 376
- Sequence property
 - about 226
 - Control object type, about 466
 - Dock Object object type 316
 - Dock Object Visibility Rule object type 317
 - form applet control property 535
 - List Column object type, about 476
 - static picklist object list 643
- Series Field property
 - By field, about populating 747
 - Chart object type 741
- server database
 - ddlsync.ksh, executing 284
 - mobile users, propagating changes to 289
 - schema extensions, applying and activating changes 281
 - schema extensions, preparing before applying 280
- server views, note, about not modifying 488
- servers, deployed on components 77

- Show picklist
 - about 743
 - configuring, properties and examples 744
- Siebel 2000, extension tables created
 - in 260
- Siebel Anywhere
 - and MLOV-enabled fields 683
- Siebel Anywhere, about using to upgrade
 - mobile clients 951
- Siebel Applet Designer
 - about 92
- Siebel Application Upgrader 155
 - repositories, about using to upgrade 966, 967
- Siebel Assignment Manager, configuring to use MLOV-enabled fields 681
- Siebel base tables
 - note, LONG extension columns, about adding to 268
- Siebel Business Process Designer
 - about using 187
- Siebel BusObject Designer 91
- Siebel Cascading Style Sheets
 - described 74
- Siebel client, caution, installing in Siebel Tools directory 33
- Siebel Code Generator Business Service,
 - about code generated by 86
- Siebel Database Extension Designer 155
- Siebel Database Server
 - pre-install tasks, about 32
- Siebel Enterprise Integration Manager. *See* EIM interface tables
- Siebel Field Service and MLOV-enabled fields 683
- Siebel Remote
 - See* dock object
 - integrating with after implementing a database extension 254
- Siebel repository. *See* repository
- Siebel Spell Check, configuring 597
- Siebel tags
 - about 796
 - controls and IDs, mappings between, about and process (diagram) 796
 - described 74
 - nesting, about and example 800
 - singleton and multi-part tags, about and example 797
 - "This" tag, described 798
- Siebel templates
 - See also* Siebel tags
 - described 74
 - interface, process of creating 791
 - templates, list of types 793
 - understanding 789
 - Web preferences, setting 795
- Siebel Tools
 - about 89
 - application window 90, 91
 - installing 32
 - using the repository validator to detect errors in configuring .COM applications 170
- Siebel Tools components
 - Application Upgrader 155
 - BusObject Repository Manager 98
 - New Object wizards 93
 - Siebel BusObject Designer 91
- Siebel Tools Object Explorer window, using
 - to view project by object definitions 992, 993
- Siebel upgrade inheritance, about 156
- Siebel Web Client, about high interactivity 80
- Siebel Web Engine
 - about 77
 - architecture and functionality described 78
 - changing the order of precedence in the .SWF file 908
 - Web application, about generating 79
 - Web application, running 79, 80
- Siebel Web Format (.SWF) extension,
 - described 794

Siebel Web installation
 directories, location and contents 913
 Siebel Workflow Manager, configuring for
 MLOV-enabled fields 676
 Siebel-defined columns, note, using
 extension columns 218
 Single Value Field
 object type, viewing 435
 properties 434
 Single Value field properties, list of 434
 singleton tags, about and example 797,
 798
 Single-Value Field properties, list of 534
 Size, Chart Element object type 750
 sizing, chart applet 751
 sort
 application, adding sorting capabilities
 to 913, 915
 Sort Specification property
 about, example. and syntax 347
 Business Component object type 432
 default value 348
 Source (primary key) field, about 397
 source control integration
 Check in/Out tab, about and
 options 965, 966
 Source Field property
 Join object type 384
 Join Specification object type 383
 Link object type, about based on 397
 spell check, configuring 597, 600
 srcctrl.bat batch file
 about and arguments (table) 959, 960
 DOS window launched, purpose of 958
 program code, about changing 963
 source code example 960
 variables, table of 962
 Windows 95 and 98, MS-DOS
 limitations 963
 srvmg.exe utility, using to regenerate the
 template local database 290
 stand alone table
 note, invoking Dock Wizard from 327
 stand-alone table, about 292
 Standard (Static) Extensibility
 about 251
 Standard Database Extensibility
 components, listed 252
 techniques to fulfill business
 requirements, without using
 Advanced Database
 Extensibility 254
 standard extension columns
 about and table of 220
 described 61
 name in extension tables 219
 standard extension tables, described 198
 Starting With text box, in association
 applets 711
 state model
 about using 188
 static database extensions, about and list of
 types 253
 static drilldown, about and configuration
 (diagram) 495
 static picklists
 about, example, and comparison with
 pick applet 638
 architecture (diagram) 640
 object definitions, list of 641
 object types architecture (diagram) 641
 pop-up visibility rules 650
 string constants
 search specification, in a 349, 486
 strings
 attributes, finding multiple 602
 exporting 605
 importing 605, 607
 untranslated, finding and exporting 602,
 603
 Style option
 Visualization view shortcut menu 148
 style sheets, Cascading Style Sheets,
 described 74
 Subcategory View picklist
 about and screen example 809

- hierarchy of screen, categories, and categorized views (diagram) 809
- swe:subviewbar, about, tag usage, and example 811
- tools configuration 810
- Summary window, log file containing contents of 931
- SWE conditional tags, list of and attributes 800, 880, 882
- swe tags
 - tree control, for 856, 857
- SWE templates
 - Applet Web Template, object properties 828, 829
 - menu configurations 868, 871
 - search and find applet tags, list of tags and example 875, 877
 - Search/Find results list applet, list of tags and example 877, 878
 - toolbar configurations 866, 868
- SWE. *See* Siebel Web Engine
- .SWF files
 - Formatting templates extension 794
- system columns
 - system fields, about and list of 224
 - system fields, table identifying correspondence between fields and columns 224
 - values and common types described 222
- system columns, described 61
- system data sources
 - creating 40
- system fields, about and field and columns, table of 366
- System value
 - Type property of Column object type 218
 - Type property of Index object type 226

T

- T suffix extension table names. *See* extension tables

- T suffix table names. *See* TAS tables
- tab bar
 - described 803
 - swe:screenbar, about and code snippet 803, 804
 - tools configuration 803
- table object definitions
 - described 62
 - list of 195
- Table property
 - Business Component object type 432
 - Join object type 383
 - joined table 384
- Table Wizard
 - columns, created by 297
 - note, types of tables created 293
 - using 293
- tables
 - See also* extension tables
 - Advanced Database Extensibility, list that can be added using 292
 - base tables, two different contexts used in 194
 - data object types, described 62
 - Data Objects layer, described and list of 60, 61
 - data tables, about 196
 - database, about configuring EIM and docking/routing interfaces 228
 - extension table, standard and custom extension tables, distinctions between 197
 - joined, about 384
 - note, modifying user keys 227, 232
 - table object definition and standard tables described 193
 - table object definitions, list of 195
- Tables report
 - described 144
 - S_ORG_EXT table, getting report for 143
- Tables version, Relationship Visualization View
 - example and described 151

- navigating within 151
- tags
 - Siebel tags, described 74
- Target Browser Group combo box, using to group browsers 888
- target columns
 - defined 665
 - display values vs. language-independent codes 674
 - LOV Type and multilingual support 663
 - Multilingual property 663
 - multilingual use, enabling for 663, 664
 - translating display values 664
 - upgrading bounded columns 665
- target property, valid properties in the picklist 586
- tasks
 - configuration 164
- template, configuration features
 - browser group specific templates, designing 884
 - .CIF files, about adding 913
 - control layout and formatting, customizing 903
 - custom HTML type, about SWE using 904
 - directories, location and contents 913
 - error messages within form, displaying 911
 - new HTML type, creating and using 903
 - sorting capabilities, adding to the application 913
 - SWE conditional tags 800, 880
- templates
 - Siebel template, described 74
 - Web layout, changing to another template 567
- Territory Assignment and MLOV, about implementation 674
- Territory Assignment Manager
 - about using 187
- test environment, repository naming convention guidelines 43
- testing with browsers 170
- Text Alignment property
 - List Column object type, about 477
- Text Alignment property, form applet control property 535
- text boxes
 - characteristics 466
 - properties 467
- Text Length property, Single Value Field object type 434
- Text Style object type 453
- TextBox type controls
 - control behavior, enhancing 540
- "This" tag, described 798
- Thread Applet property, View object definition 494
- Thread Bar
 - about, example, and properties 493
 - configuring 494
- thread bar
 - physical navigation, about 871
- Thread Field property, View object type 494
- Thread Title property, View object type 494
- time control, list applet type property 485
- Time data type, Physical Type of Column object type 217
- time values, using in text box 468
- time, and date, setting 135
- Title property, Single Value Field object type 534
- Title value, Chart Element object type 750
- toggle applets
 - about and creating 849
- toolbars
 - about and figures 513
 - menu items and toolbars, activating and suppressing 520
 - method targeting, invoking 521
 - new, defining new toolbar 584
 - SWE templates, toolbar configuration 866

- thread bar, about physical navigation 871
 - toolbar and menu-related object types 515
 - toolbar configuration, about 866
 - toolbar icon, adding to existing toolbar 585
 - Total Required property, List Column object type 477
 - Translatable column, described 684
 - translatable types, defined 659
 - Translate property, about using 660
 - translations
 - and LOV Type 659
 - custom extension columns 658
 - display values, adding 664
 - mode, about running MLOV Upgrade Utility 665
 - Tree Applet Wizard, using to create a new tree applet 763
 - tree applets
 - about and example 460
 - about and examples 756
 - about and screen example 853
 - Applet Web Template Layout Window, using and options 764
 - configuration file parameters, tree control contents 858
 - configuring, about and object types 760
 - recursive trees, about and properties 764
 - tree control, swe tags 856
 - using Tree Applet Wizard to create 763
 - Tree Node object type, about and properties 761, 762
 - Tree object type, about 761
 - Type field, about 362
 - Type picklists, about and default type 742
 - Type property
 - Column object type, about 217
 - Control object type, about 465
 - extension columns, about adding to form applet control property 535
 - Index object definition, about 226
 - Single Value Field object type 434
 - Types tab
 - described and example 101
- ## U
- UI elements
 - using Web Client Migration Wizard to the Web client 968
 - UI objects
 - copying, guidelines for 973
 - unbounded picklists
 - note, typing new values into 630
 - Undo Record object definitions, undoing 123
 - undocking windows
 - preventing when moved 113
 - procedure 113
 - undoing project check out 1009
 - Unique property, about 226
 - Upgrade Ancestor, viewing all objects marked as 145
 - Upgrade Ancestor, viewing objects marked as
 - See upgrading inheritance
 - Upgrade Inheritance property
 - caution, creating new object definitions 121
 - parent object, list of all descendents set to 152
 - upgrade utility, about running in two modes 665
 - upgrading
 - caution, creating objects not automatically upgraded 121
 - upgrading inheritance
 - about
 - copy of object, creating 973
 - copying objects, guidelines 970
 - descendent or copies of an object, viewing 974
 - enhancements, process of applying 972

- parent to descendents, propagating changes from 975
 - repositories, about comparison of 972
 - UI objects and business components, guidelines for copying 973
 - using scenario 970
 - Use Primary Join property, about 414
 - user interface
 - physical, described 74
 - standard Siebel 159
 - user interface object types
 - architectural diagrams of 338
 - user interface objects
 - architecture (diagram) 450
 - business objects, mapping (diagram) 443
 - object types, list of 451
 - screen example 448
 - User Interface Objects layer
 - See also* Physical User Interface layer
 - applets, described and example 69, 70
 - applications, described and example 72
 - interface elements diagram 68
 - logical UI object types, described 72, 73
 - screens, described and example 71
 - views, described 71
 - user interface objects, configuring
 - See also* application; list applets
 - applets, about 530
 - bitmap button controls, about using in applets and views 584
 - bitmap category, creating and adding a bitmap to it 583
 - bitmaps, about associating with applets, toolbar icons, and view bars 582
 - form applet control properties 535
 - HTML Type property 540
 - new view, about providing access to 570
 - process steps 529
 - screen view, defining 574
 - screens, about defining 572
 - screens, defining procedure 573
 - toolbar icon, adding to an existing toolbar 585
 - toolbar, defining a new toolbar 584
 - views, about defining 562
 - views, defining using the Object List Editor 564
 - User Key Attribute Join, about 236
 - User Key Attribute, about 236
 - User Key Column, about 236
 - User Key object type, about 236
 - User Key value, Type property of Index object type 226
 - user keys, note, modifying in Siebel table or EIM base tables 227, 232
 - User Name property, about 196
 - User Profile Attributes, object used by personalization for user profile 185
 - UTC Date Time, Physical Type of Column object type 217
- V**
- validate
 - object definitions procedure 123
 - Validate window user interface elements 125, 126
 - validation options, changing 126
 - Validate window
 - user interface elements 125, 126
 - validating procedure 123
 - validation options, changing 126
 - validation mode, MLOV Upgrade Utility 665
 - validation mode, MLOV Upgrade Utility, about 665
 - Value property
 - Business Component object type 375
 - values
 - repicking (Workflow Policy Column) 678
 - repicking (Workflow Policy Program Argument) 680
 - Varchar data type, Physical Type of Column object type 216

- VB functionality, about writing VB code 675
 - view
 - object type, described 73
 - user interface element, described 71
 - view (Opportunity Detail-Contacts List view) object type, about 709
 - View Hierarchy diagram, navigating within 154
 - View object, Tree applet properties 760
 - View template
 - template type 793
 - View templates
 - HTML frames, using in 825, 826, 828
 - View Web template item
 - described 73
 - user interface type 453
 - View Web template, described 73
 - View Web template, user interface type 452
 - View Wizard, using to create view 563
 - view, user interface type 451
 - views
 - See also* drilldown behavior
 - about and styles of 488
 - about defining 562
 - bitmap button controls, about using in 584
 - creating using the View Wizard 563, 564
 - new view. about providing access to 570
 - Object List Editor, defining view in 564
 - Web layout, adding applet to 566
 - Web layout, editing using the Web Layout Editor 566
 - virtual business components
 - about 358
 - Virtual Table value, about 195
 - visibility
 - properties related to 182
 - Visibility picklist
 - about and rules for context views for a selected view 804
 - SWE template configuration, about and code syntax 806
 - visibility rules
 - checking 659
 - MLOVs, and configuring 658
 - visibility strength concept in dock object visibility 317
 - Visibility Strength property
 - Dock Object object type 316
 - Dock Object Visibility Rule object type 317
 - Visual SourceSafe
 - examples 964
 - repository management, about using in 957
 - Visual Web Page Editor
 - Web Page Layout Editor, accessing 579, 582
 - visualization diagram, selecting object definition properties 148
 - Visualization View
 - Descendents Visualization View 152
 - Details Visualization View 146
 - invoking, two methods of 145
 - list of views 145
 - Relationships Visualization View 149
 - Web Hierarchy Visualization View 153
- ## W
- W prefix table names, about 194
 - Warehouse value, about 195
 - Web application
 - personalizing 185
 - running 79, 80
 - Web applications, running
 - HTML frames, about and screen example 817
 - HTML frames, about using in View templates 825
 - HTML frames, support for multiple views on a page 822
 - HTML frames, swe template configuration 819

- HTML frames, using container page template 820
 - HTML frames, using in View templates 826
 - Web Page Layout Container Page, about 816
 - Web Client architecture
 - JavaScript objects, described 81
 - Web application, about generating 79
 - Web application, running 79
 - Web Client Migration Wizard, using 968
 - Web client, Win32 UI elements, converting 968
 - Web Hierarchy Visualization View 153
 - described 145
 - View Hierarchy diagram, navigating within 154
 - Web layout
 - applet, adding to the Web layout of the view 566
 - template, changing to another 567
 - view design, previewing 566
 - views, editing in the Web Layout Editor 566
 - Web Layout Editor, using to edit Web layout of a view 566
 - Web Layout Editor, using to edit Web layout of applets 547
 - Web Menu Editor
 - about using 591
 - applet menus, creating 593
 - Web page
 - Web-related object shown on a Web page 509
 - Web-related object, parameter of the Web Page item 509
 - Web Page Layout Container Page
 - about 816, 817
 - Web Page Layout Editor
 - accessing 579
 - Web page objects
 - Web Page Layout Editor, accessing 579, 582
 - Web page template
 - template type 793
 - Web-related object, defines Web page template attributes 509
 - Web preferences, setting 795
 - Web template
 - See also* Web page; Web page template described 73
 - user interface type 452
 - Web Templates windows
 - described 111, 794
 - widget text box, about popping up list 467
 - wildcard characters
 - search specification, in a 349, 487
 - Win32 client, using Web Client Migration Wizard to convert UI elements 968
 - windows
 - docking, preventing when moved 113
 - docking/undocking 113
 - hiding 113
 - Windows 95 and 98, limitations and the srcctrl.bat batch file 963
 - Wizard style popup, about and example 774
 - wizards
 - Business Component Wizard, using 444
 - Workflow Policy Column
 - configuring 678
 - LIC picklist, creating for 677
 - repicking 678
 - Workflow Policy Objects report, described 144
 - Workflow Policy Program Argument
 - configuring 680
 - LIC picklist, creating for 679
 - repicking values 680
 - workflows
 - about using 187
 - Workload Rules, configuring 682
- X**
- X axis, about limiting and sorting 749
 - X suffix

-
- extension table name and examples 260
 - X suffix extension table names. *See* extension tables
 - X suffix table names. *See* one-to-one extension tables
 - X suffix, in table names. *See* extension tables
 - x symbol, displaying in a form applet 473
 - XM suffix
 - extension table name and examples 260
 - XM suffix able names. *See* one-to-many extension tables
 - XM suffix extension table names. *See* extension tables
 - XMIF suffix interface table names. *See* EIM interface tables
- Y**
- Y axis
 - multiple line curves plotted against, object definitions 748
 - note, about charts with two Y axes 721
 - two Y axes charts, configuring and properties 749
- Z**
- Z axis, about limiting and sorting 749
 - Zoom option
 - Visualization view shortcut menu 148

