Siebel

Integration Platform Technologies: Siebel Enterprise Application Integration Guide

July 2021

Siebel
Integration Platform Technologies: Siebel Enterprise Application Integration Guide

July 2021

Part Number: F12751-06

Copyright © 2021, Oracle and/or its affiliates.

Authors: Siebel Information Development Team

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display in any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or “commercial computer software documentation” pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government’s use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

The business names used in this documentation are fictitious, and are not intended to identify any real companies currently or previously in existence.

Contents

Preface i

1 What's New in This Release 1

What's New in Integration Platform Technologies: Siebel Enterprise Application Integration Guide, Siebel CRM 21.7 Update 1
What's New in Integration Platform Technologies: Siebel Enterprise Application Integration Guide, Siebel CRM 21.4 Update 1
What's New in Integration Platform Technologies: Siebel Enterprise Application Integration Guide, Siebel CRM 20.1 Update 1
What's New in Integration Platform Technologies: Siebel Enterprise Application Integration Guide, Siebel CRM 19.9 Update 2
What's New in Integration Platform Technologies: Siebel Enterprise Application Integration Guide, Siebel CRM 19.7 Update 2
What's New in Integration Platform Technologies: Siebel Enterprise Application Integration Guide, Siebel CRM 19.3 Update 2
What's New in Integration Platform Technologies: Siebel Enterprise Application Integration Guide, Siebel CRM 19.1 Update 3

2 Integration Objects 5

Integration Objects 5
About Integration Object Terminology 5
About Integration Objects 6
About Integration Object Base Object Types 7
About the Difference Between Integration Objects and Integration Object Instances 7
About Integration Object Wizards 8
About the Structure of Integration Objects 9
About Integration Component User Properties as Operation Controls 20
About EAI Siebel Adapter Access Control 27

3 Creating and Maintaining Integration Objects 29

Creating and Maintaining Integration Objects 29
About the Integration Object Builder 29

About the EAI Siebel Wizard Business Service 30
Process of Creating Integration Objects 31
Creating Integration Objects Using the EAI Siebel Wizard Business Service 31
Creating an Integration Object Based on Another Root Business Component 34
Creating an Integration Object with Many-To-Many Relationships 34
Creating Integration Object Instances Programmatically 35
Guidelines for Configuring Integration Objects 37
Validating Integration Objects 37
Testing Integration Objects 38
Deploying Integration Objects to the Run-Time Database 38
About Synchronizing Integration Objects 39
Synchronizing Integration Objects 44
Resolving Synchronization Conflicts for Integration Objects and User Properties 47
Using Formatted Values in Integration Objects 50
Generating Integration Object Schemas 50
Optimizing the Performance of Integration Objects 51
About Business Component Restrictions for Integration Components 52
Guidelines for Using Integration Components 52

4 Business Services 55

Business Services 55
About Business Services 55
Creating Business Services in Siebel Tools 58
Creating Business Services in the Siebel Application 60
Deploying Business Services as Web Services 61
Exporting and Importing Business Services in Siebel Tools 62
Importing Business Services into Siebel CRM 62
Testing Your Business Service in the Simulator 63
About Accessing a Business Service Using Siebel eScript or Siebel VB 63
Business Scenario for the Use of Business Services 64

5 Web Services 67

Web Services 67
About Web Services 67
About RPC-Literal and DOC-Literal Bindings 68
About One-Way Operations and Web Services 69
Invoking Siebel Web Services Using an External System 69

Consuming External Web Services Using Siebel Web Services 78
Using the Local Business Service 86
Examples of Invoking Web Services 90
About Web Services Security Support 96
About WS-Security UserName Token Profile Support 97
Proxy Configuration for Java Web Container 98
About Siebel Authentication and Session Management SOAP Headers 99
About Web Services and Web Single Sign-On Authentication 106
About SOAP Fault Schema Support 107
About Custom SOAP Filters 110
About EAI File Streaming 112
About Web Services Cache Refresh 114
Enabling Web Services Tracing 115
Previewing the Repository Changes Before Delivery 116

6 EAI Siebel Adapter Business Service 119
EAI Siebel Adapter Business Service 119
About the EAI Siebel Adapter Business Service 119
EAI Siebel Adapter Business Service Methods 120
About Using Effective Dating with Siebel EAI Adapter Business Service 145
Enabling Effective Dating on Fields 146
Enabling Effective Dating on Links 148
About Using Language-Independent Code with the EAI Siebel Adapter Business Service 151
Siebel EAI and Run-Time Events 152
Guidelines for Using the EAI Siebel Adapter Business Service 152
Troubleshooting the EAI Siebel Adapter Business Service 153
Enabling Logging for the EAI Siebel Adapter Business Service 153
Enabling Siebel Argument Tracing 155
Configuring the EAI Siebel Adapter Business Service for Concurrency Control 155

7 EAI UI Data Adapter Business Service 161
EAI UI Data Adapter Business Service 161
About the EAI UI Data Adapter Business Service 161
EAI UI Data Adapter Business Service Methods 162
EAI UI Data Adapter Business Service Method Arguments 177

8 Siebel Virtual Business Components 179
Siebel Virtual Business Components 179
About Virtual Business Components 179
Using Virtual Business Components 181
XML Gateway Service 183
Examples of the Outgoing XML Format 185
Search-Spec Node-Type Values 188
Examples of the Incoming XML Format 189
External Application Setup 191
Custom Business Service Methods 191
Custom Business Service Examples 204

9 Siebel EAI and File Attachments 213
Siebel EAI and File Attachments 213
About File Attachments 213
Exchanging Attachments with External Applications 213
Using MIME Messages to Exchange Attachments 214
About the EAI MIME Hierarchy Converter 220
About the EAI MIME Doc Converter 222
Using Inline XML to Exchange Attachments 224

10 External Business Components 229
External Business Components 229
Process of Configuring External Business Components 229
Using Specialized Business Component Methods for EBCs 241
Usage and Restrictions for External Business Components 242
About Using External Business Components with the Siebel Web Clients 243
About Overriding Connection Pooling Parameters for the Data Source 244
About Joins to Tables in External Data Sources 244
About Distributed Joins 245
Troubleshooting External Business Components 246

11 Predefined EAI Business Services 249
Predefined EAI Business Services 249
Predefined EAI Business Services 249

12 Property Set Representation of Integration Objects 253

Property Set Representation of Integration Objects 253
Property Sets and Integration Objects 253
Example Instance of an Account Integration Object 255

13 DTDs for XML Gateway Business Service 257

DTDs for XML Gateway Business Service 257
Outbound DTDs for the XML Gateway Business Service 257
Inbound DTDs for the XML Gateway Business Service 259

Preface

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

Using Oracle Applications

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

Documentation Accessibility

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

Contacting Oracle

Access to Oracle Support

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

Comments and Suggestions

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

1 What's New in This Release

What's New in Integration Platform Technologies: Siebel Enterprise Application Integration Guide, Siebel CRM 21.7 Update

No new features have been added to this guide for this release. The following topic was revised to improve the technical accuracy of this guide:

- *About Siebel Authentication and Session Management SOAP Headers*

What's New in Integration Platform Technologies: Siebel Enterprise Application Integration Guide, Siebel CRM 21.4 Update

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

Topic	Description
Configuring Siebel Server and Config Agent for SHA2 Outbound	Modified topic. Updated the content to provide details about the EAIOutboundSubSys component parameter.
Usage and Restrictions for External Business Components	Modified topic. All EBCs require the Siebel S_APP_VER and S_SYS_PREF tables to be present in the external database.

What's New in Integration Platform Technologies: Siebel Enterprise Application Integration Guide, Siebel CRM 20.1 Update

No new features have been added to this guide for this release. This guide has been updated to reflect only product name changes.

What's New in Integration Platform Technologies: Siebel Enterprise Application Integration Guide, Siebel CRM 19.9 Update

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

Topic	Description
Migrating Outbound Web Services	New topic. It provides details to migrate outbound web services

What's New in Integration Platform Technologies: Siebel Enterprise Application Integration Guide, Siebel CRM 19.7 Update

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

Topic	Description
Supported Web Services Standards *RPC-Literal Support* *DOC-Literal Support* *Defining Support for One-Way Operations*	Modified topics. Removed references to Web services standards supported with Siebel CRM to reference Certifications tab in My oracle Support.

What's New in Integration Platform Technologies: Siebel Enterprise Application Integration Guide, Siebel CRM 19.3 Update

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

Topic	Description
Previewing the Repository Changes Before Delivery	New topic. Added one case for querying with the `Workspace&Version` flag to illustrate preview of metadata changes in a workspace.

What's New in Integration Platform Technologies: Siebel Enterprise Application Integration Guide, Siebel CRM 19.1 Update

No new features have been added to this guide for this release. This guide has been updated to reflect only product name changes.

2 Integration Objects

Integration Objects

This chapter describes the structure of Siebel integration objects. It describes the Integration Object Builder wizard, which assists you in building your own integration objects based on Siebel objects.

The chapter consists of the following topics:

- *About Integration Object Terminology*
- *About Integration Objects*
- *About Integration Object Base Object Types*
- *About the Difference Between Integration Objects and Integration Object Instances*
- *About Integration Object Wizards*
- *About the Structure of Integration Objects*
- *About Integration Component User Properties as Operation Controls*
- *About Integration Component Keys*
- *About EAI Siebel Adapter Access Control*

About Integration Object Terminology

This chapter describes concepts that are often referred to using inconsistent terminology on different systems. The following table has been included to clarify the information in this chapter by providing a standard terminology for these concepts.

Term	Description
Component	A constituent part of any generic object.
Field	A generic reference to a data structure that can contain one data element.
Integration message	A bundle of data consisting of two major parts: • Header information that describes what will be done with or to the message itself • Instances of integration objects; that is, data in the structure of the integration object
Integration object	An integration object of any type, including the Siebel integration object, the SAP BAPI integration object, and the SAP IDOC integration objects.
Integration object instance	Actual data, usually the result of a query or other operation, which is passed from one business service to another, that is structurally modeled on a Siebel integration object.

Term	Description
Metadata	Data that describes data. For example, the term data type describes data elements such as char, int, Boolean, time, date, and float.
Siebel business component	A Siebel object type that defines a logical representation of columns in one or more database tables. A business component collects columns from the business component's base table, its extension tables, and its joined tables into a single structure. Business components provide a layer of abstraction over tables. Applets in Siebel applications reference business components; they do not directly reference the underlying tables.
Siebel business object	A Siebel object type that creates a logical business model using links to tie together a set of interrelated business components. The links provide the one-to-many relationships that govern how the business components interrelate in this business object.
Siebel integration object	An object stored in the Siebel repository that represents a Siebel business object.
Siebel integration component	A constituent part of a Siebel integration object that represents a Siebel business component.
Siebel integration component field	A data structure that can contain one data element in a Siebel integration component. Represents a Siebel business component field.

About Integration Objects

Siebel integration objects allow you to represent integration metadata for Siebel business objects, XML, and other external data structures as common structures that the Enterprise Application Integration (EAI) infrastructure can understand. Because these integration objects adhere to a set of structural conventions, they can be traversed and transformed programmatically, using Siebel eScript objects, methods, and functions, or transformed declaratively using Siebel Data Mapper.

> **Note:** For more information on Siebel Data Mapper, see *Business Processes and Rules: Siebel Enterprise Application Integration* .

The typical integration project involves transporting data from one application to another. For example, if you want to synchronize data from a back-office system with the data in your Siebel application. You might want to generate a quote in the Siebel application and perform a query against your Enterprise Resource Planning (ERP) system transparently. In the context of Siebel EAI, data is transported in the form of an *integration message*. A message, in this context, typically consists of header data that identifies the message type and structure, and a body that contains one or more instances of data-for example, orders, accounts, or employee records.

When planning your integration project, consider several issues:

- How much data transformation does your message require?
- At what point in the process do you perform the data transformation?
- Is a confirmation message response to the sender required?

- Are there data items in the originating data source that will not be replicated in the receiving data source, or that will replace existing data in the receiving data source?

This guide can help you understand how Siebel EAI represents the Siebel business object structure. It also provides descriptions of how Siebel EAI represents external SAP R/3 structures.

About Integration Object Base Object Types

Each integration object created in Siebel Tools has to be based on one of the base object types presented in the following table. This property is used by adapters to determine whether the object is a valid object for them to process.

Note: XML converters can work with any of the base object types.

Base Object Type	Description
None	For internal use only.
SQL	Used for manually creating integration objects. Only the EAI SQL Adapter accepts integration objects of this type.
SQL Database Wizard	Used by the Database Wizard for the integration object it creates. Only the EAI SQL Adapter accepts integration objects of this type.
SQL Oracle Wizard	Used by the Oracle Wizard for the integration object it creates. Only the EAI SQL Adapter accepts integration objects of this type.
Siebel Business Object	Used by the Integration Object Builder wizard for the integration object it creates. The EAI Siebel Adapter accepts only the integration object of this type.
Table	Obsolete.
XML	Used to represent external XML Schema such as DTD or XSD. For information on DTD and XSD, see *XML Reference: Siebel Enterprise Application Integration* .

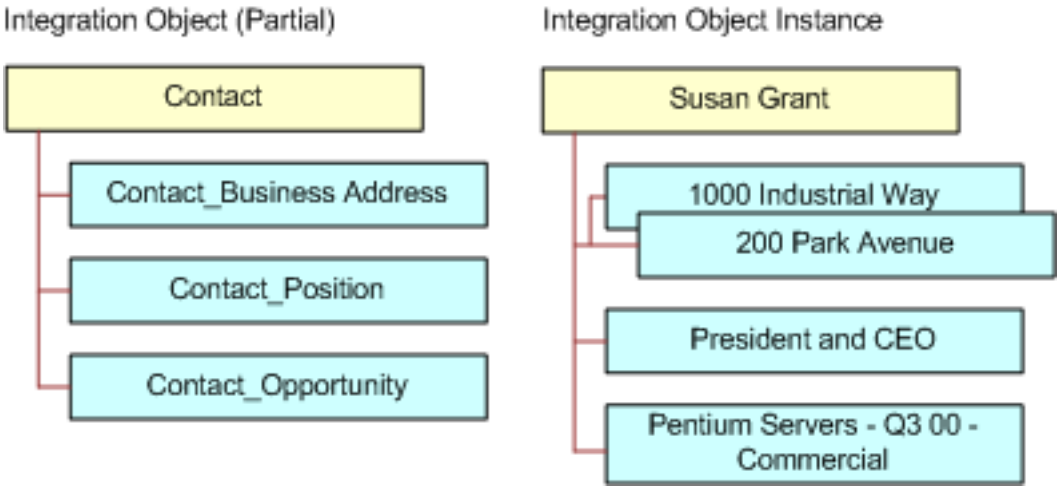
About the Difference Between Integration Objects and Integration Object Instances

Understanding the difference between integration objects and integration object instances is important, especially in regard to the way they are discussed in this chapter.

An integration object, in the context of Siebel EAI, is metadata; that is, it is a generalized representation or model of a particular set of data. It is a schema of a particular entity.

An integration object instance is also referred to as a Siebel Message object.

An integration object instance is actual data organized in the format or structure of the integration object. The following figure illustrates a simple example of an integration object and an integration object instance, using partial data.



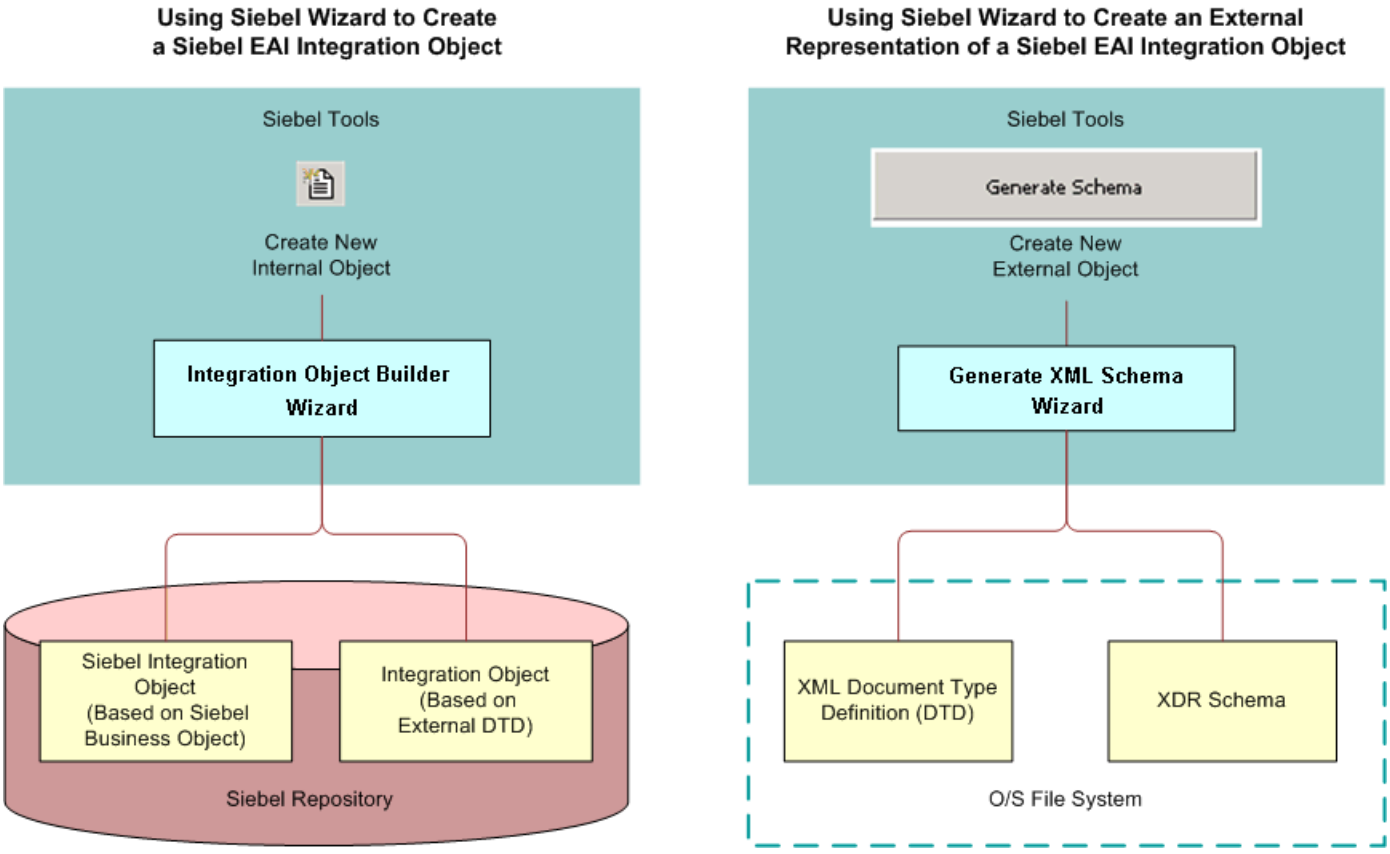
Any discussion of integration objects in this book will include clarifying terms to help make the distinction, for example, metadata or Siebel instance.

About Integration Object Wizards

Within Siebel Tools, there are multiple wizards associated with integration objects:

- One that creates integration objects for internal use by the Siebel application
- Others that create integration objects for external systems based on Siebel objects

The following figure shows the logic of the Integration Object Builder and Generate XML Schema wizards. The Code Generator wizard (not shown) works in the same manner as the Generate XML Schema wizard, but it generates Java classes.



The following are the integration object wizards:

- **Integration Object Builder wizard.** This wizard lets you create a new object. It supplies the functionality for creating integration objects from Siebel business objects or integration objects, based on the representations of external business objects using XML Schema Definition (XSD) or Document Type Definition (DTD). Access this wizard from the New Object Wizards dialog box in Siebel Tools. After selecting the EAI tab, double-click Integration Object to start the Integration Object Builder wizard.
- **Generate XML Schema wizard.** This wizard lets you choose an integration object and output XML schema in XML Schema Definition (XSD) standard, Document Type Definition (DTD), or Microsoft's XDR (XML Data Reduced) format. In the Integration Objects list in Siebel Tools, select an integration object. Then click Generate Schema to start the Generate XML Schema wizard.
- **Code Generator wizard.** The third wizard lets you create a set of Java class files based on any available integration object or Siebel business service. In the Integration Objects list in Siebel Tools, select an integration object. Then click Generate Code to start the Code Generator wizard.

Note: Specific instructions on how to use these wizards appear throughout the Siebel Enterprise Application Integration documentation set where appropriate.

About the Structure of Integration Objects

The Siebel integration object provides a hierarchical structure that represents a complex data type. Most specifically, prebuilt EAI integration objects describe the structure of Siebel business objects, SAP IDOCs, SAP BAPIs, XML, and

external data. Most integration projects require the use of an integration object that describes Siebel business objects, either in an outbound direction such as a *query* operation against a Siebel integration object, or in an inbound direction such as a *synchronize* operation against a Siebel integration object.

Creating and Maintaining Integration Objects describes how to create integration objects. The initial process of using the Integration Object Builder wizard is essentially the same for every integration object type currently supported.

> **CAUTION:** Avoid using or modifying integration objects in the EAI Design project. Using or modifying any objects in the EAI Design project can cause unpredictable results. The best practice is to create a separate project for your integration objects, for example, ABC Integration Objects, where ABC is the name of your company.

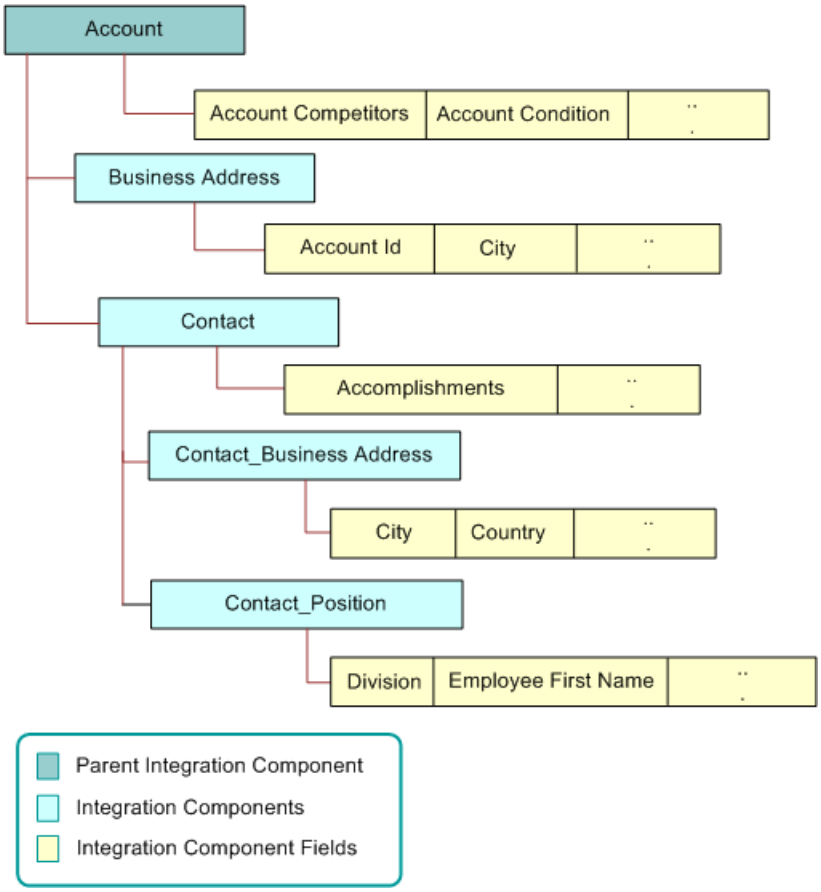
Siebel business objects conform to a particular structure in memory, although it is generally not necessary to consider this structure when working with Siebel CRM. However, when you are planning and designing an integration project, it is helpful to understand how a Siebel EAI integration object represents that internal structure.

An integration object consists of one Parent Integration Component, sometimes referred to as the root component, or the primary integration component. The Parent Integration Component corresponds to the primary business component of the business object you chose as the model for your integration object.

For example, assume you chose the Account business object (on the first panel of the Integration Object Builder wizard) to base your integration object myAccount_01 on. The Account business object in Siebel Tools has an Account business component as its primary business component. In the myAccount_01 integration object, every child component will be represented as either a direct or indirect child of the primary business component named Account.

Each child component can have one or more child components. In Siebel Tools, if you look at the integration components for an integration object you have created, then you see that each component can have one or more fields. The following figure illustrates a partial view of a Siebel integration object based on the Account business object, with the Business Address component and the Contact component activated.

The following figure represents part of the structure of the Account integration object. The Account parent integration component can have both fields and child integration components. Each integration component can also have child integration components and fields. A structure of this sort represents the *metadata* of an Account integration object. You can choose to inactivate components and fields. By inactivating components and fields, you can define the structure of the integration object instances entering or leaving the system.



The following topics are also discussed:

- *About the Cardinality of Child Integration Components*
- *Custom Integration Component Fields*
- *Integration Components and Associations*
- *Multivalue Groups Within Business Components*
- *Setting Primaries Through Multivalue Links*
- *Validation of Integration Component Fields and Picklists*
- *Calculated Fields and Integration Objects*
- *Inner Joins and Integration Components*
- *Defining Field Dependencies*
- *Repository Objects*
- *About Integration Component User Properties as Operation Controls*
- *About Integration Component Keys*

About the Cardinality of Child Integration Components

By default each child integration component created in Siebel Tools is assigned a cardinality value of *Zero or More*. The values *Zero or More*, or *Zero or One*, mean that the corresponding integration component is optional. Setting the value to *One*, or *One or More*, means at least one integration component instance must be included in the hierarchy. When the

cardinality value is set to *One*, you must have one and only one instance of the integration component in the hierarchy. The following table information lists possible cardinality values.

Cardinality	Integration Component Instance
One	An integration component is mandatory and limited to one instance.
One or more	An integration component is mandatory and must contain at least one instance.
Zero or more	An integration component is optional, and more than one instance is allowed.
Zero or one	An integration component is optional, but if there is one present, then the limit is one.

Custom Integration Component Fields

In some cases, you might want to pass custom attributes that are not necessarily part of the actual data but related to the context of the data. You can use various means such as SOAP headers and transport headers to pass such custom fields, or you can have them as part of the integration schema as custom integration component fields.

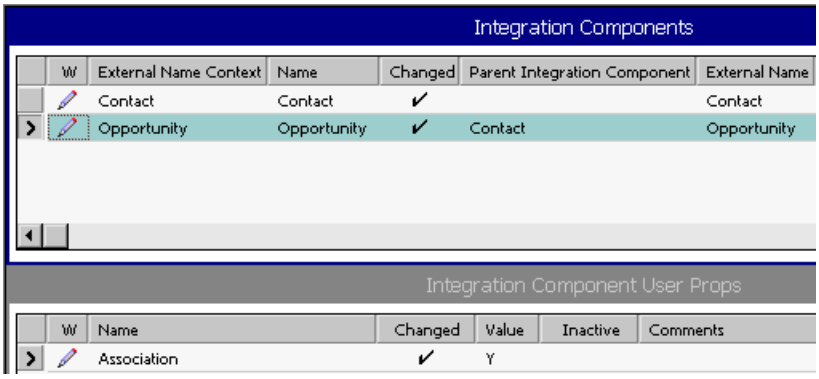
Custom attributes can be added manually to integration objects as integration component fields. The integration component field type (Type property in the Object List Editor, Field Type in the Properties window) of custom attributes must be set to Custom. XML style can be chosen as Attribute or Element, appearing in the schema as XML attributes and XML elements, respectively.

The new custom attributes will appear in the schema generated from the integration object, like any other integration component field.

Integration Components and Associations

Siebel business objects are made up of business components that are connected by a *link*. An *association* is a business component that represents the intersection table that contains these links. The integration component definition of associations is similar to that of multivalue groups (MVGs). User properties Association and MVGAssociation on the integration component denote that the corresponding business component is an associated business component or an associated MVG, respectively. For fields that are defined on MVG associations, External Name denotes the name of the business component field as it appears on the parent business component, and the user property AssocFieldName denotes the name of the business component field as it appears on the MVG business component.

For example, the Contact business object is partly made up of the Contact and Opportunity business components. The association between these two business components is represented by the Contact/Opportunity link with a value or a table name in the Inter Table column. The Integration Object Builder wizard creates a new integration component for the integration object, based on the Contact business object that represents the association. As shown in the following figure, the Opportunity integration component has one user property defined: Association, set to a value of Y.



Note: When building an integration object, if an integration component is an association based on an intersection table, then the user key for this integration component can contain fields based on the same intersection table only if the integration component has an AllowMultipleAssociations integration component user property set to Y in Siebel Tools.

Multivalue Groups Within Business Components

Multivalue groups (MVGs) are used within Siebel business components to represent database multivalue attributes. MVGs can be one of two types: regular MVGs or MVG Associations.

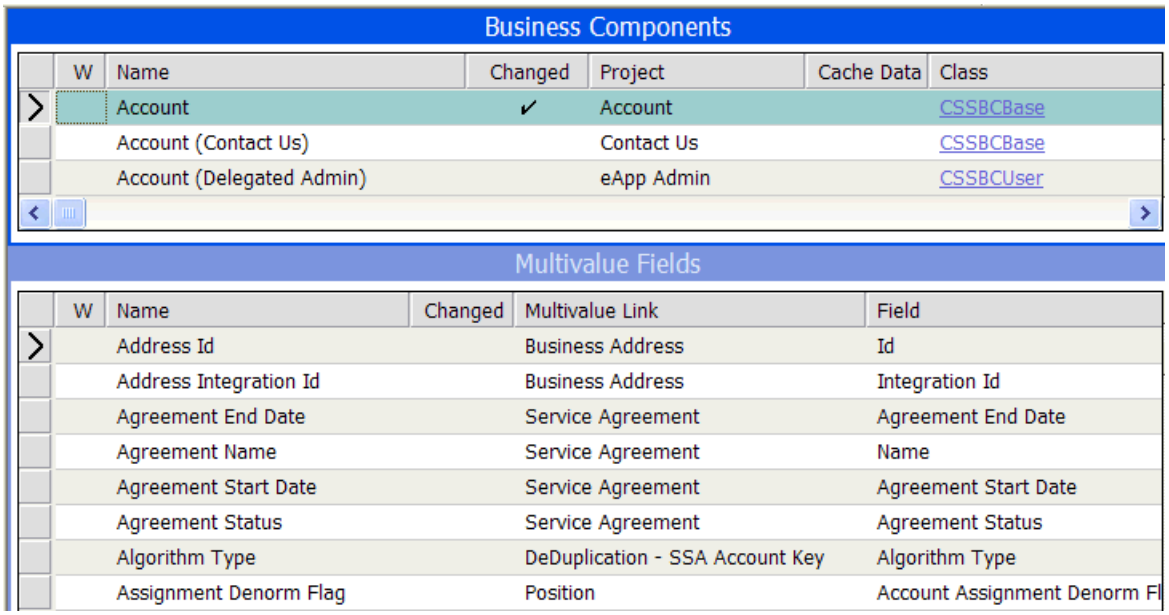
An integration object instance most often has multiple integration component instances. For example, an Account can have multiple Business Addresses but only one of these addresses is marked as the primary address. A business requirement might require that only the integration component instance that corresponds to the primary MVG be part of the integration object instance. In relation to Account and Business Addresses this means that only the primary address will be part of the Account integration object instance. The primary address can be obtained by one of the following steps:

- Creating a new MVG on the Account business component that uses a link with a search specification only returning the primary address record.
- Exposing the primary address information on the Account business component level using a join that has the primary ID as source field. Note that in this case the primary address information corresponds to fields on the Account integration component instance and not the fields on a separate Address component instance.

In Siebel Tools, if a Siebel business component contains an MVG, then the MVG is represented by several objects as illustrated in the following topics.

Multivalue Fields in a Business Component

For example, as illustrated in the following figure, the Account business component contains a multivalue field, Address Id. The multivalue link property of Address Id has the value Business Address.



Business Components

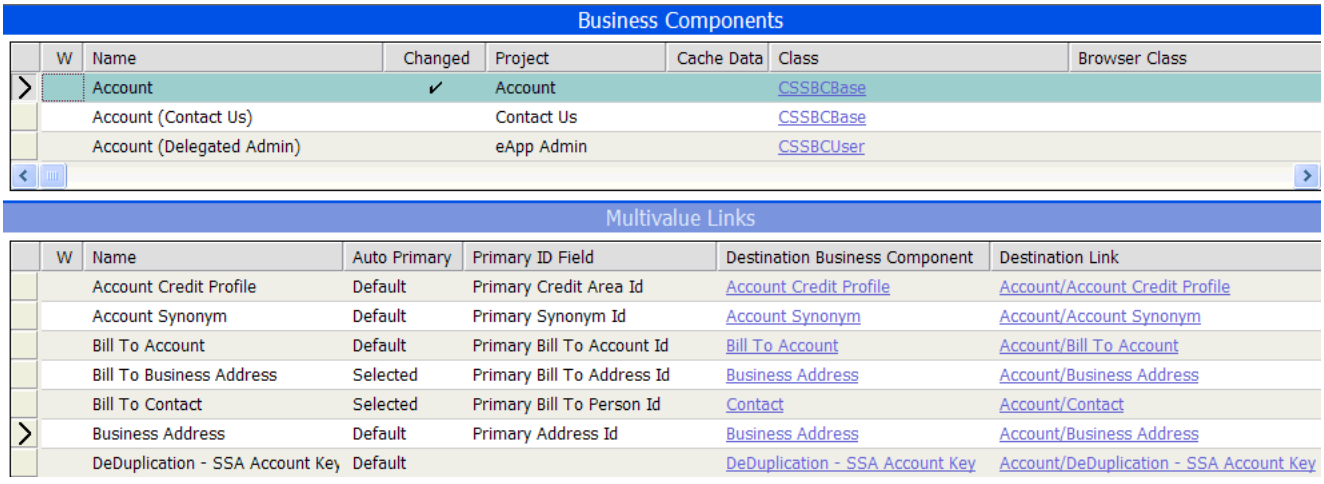
W	Name	Changed	Project	Cache Data	Class
>	Account	✓	Account		CSSBCBase
	Account (Contact Us)		Contact Us		CSSBCBase
	Account (Delegated Admin)		eApp Admin		CSSBCUser

Multivalue Fields

W	Name	Changed	Multivalue Link	Field
>	Address Id		Business Address	Id
	Address Integration Id		Business Address	Integration Id
	Agreement End Date		Service Agreement	Agreement End Date
	Agreement Name		Service Agreement	Name
	Agreement Start Date		Service Agreement	Agreement Start Date
	Agreement Status		Service Agreement	Agreement Status
	Algorithm Type		DeDuplication - SSA Account Key	Algorithm Type
	Assignment Denorm Flag		Position	Account Assignment Denorm Fl

Multivalue Links in a Business Component

The Business Address multivalue link associated with the Address Id multivalue field in the previous figure has the value Business Address as its Destination Business Component, as shown in the following figure.



Business Components

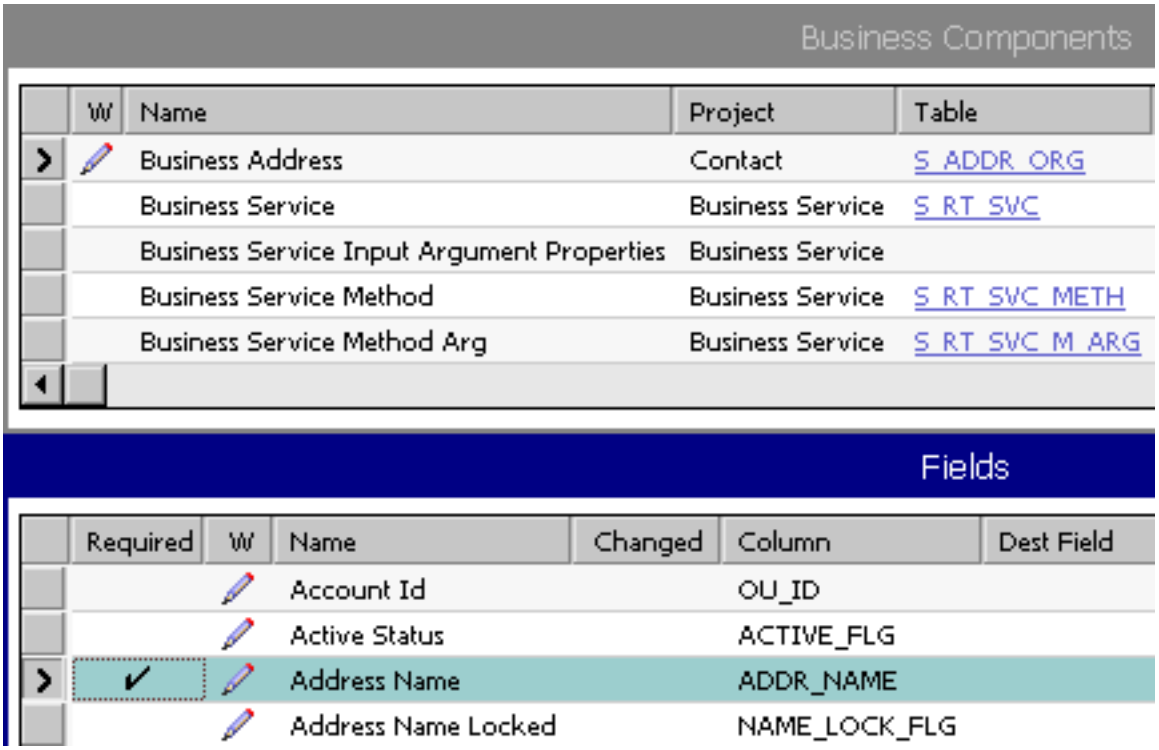
W	Name	Changed	Project	Cache Data	Class	Browser Class
>	Account	✓	Account		CSSBCBase	
	Account (Contact Us)		Contact Us		CSSBCBase	
	Account (Delegated Admin)		eApp Admin		CSSBCUser	

Multivalue Links

W	Name	Auto Primary	Primary ID Field	Destination Business Component	Destination Link
	Account Credit Profile	Default	Primary Credit Area Id	Account Credit Profile	Account/Account Credit Profile
	Account Synonym	Default	Primary Synonym Id	Account Synonym	Account/Account Synonym
	Bill To Account	Default	Primary Bill To Account Id	Bill To Account	Account/Bill To Account
	Bill To Business Address	Selected	Primary Bill To Address Id	Business Address	Account/Business Address
	Bill To Contact	Selected	Primary Bill To Person Id	Contact	Account/Contact
>	Business Address	Default	Primary Address Id	Business Address	Account/Business Address
	DeDuplication - SSA Account Key	Default		DeDuplication - SSA Account Key	Account/DeDuplication - SSA Account Key

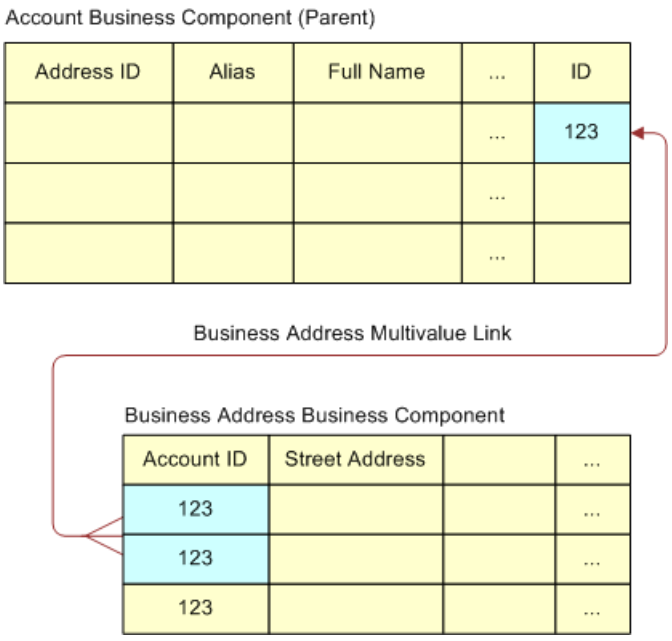
Fields in a Business Component After Adding a Multivalue Link

The fact that the Business Address multivalue link has Business Address as its Destination Business Component means that there is another business component named Business Address. The Business Address business component contains the fields that are collectively represented by Address Id in the Account business component, as shown in the following figure.



Graphical Representation of a Business Component and a Multivalue Link

The following figure shows a graphical way to represent the relationship between Account business component and the Business Address multivalue link.



The more table-like representation in the following figure shows how the Business Address multivalue link connects the two business components. The child points to the Business Address business component, which contains the multiple fields that make up the MVG.

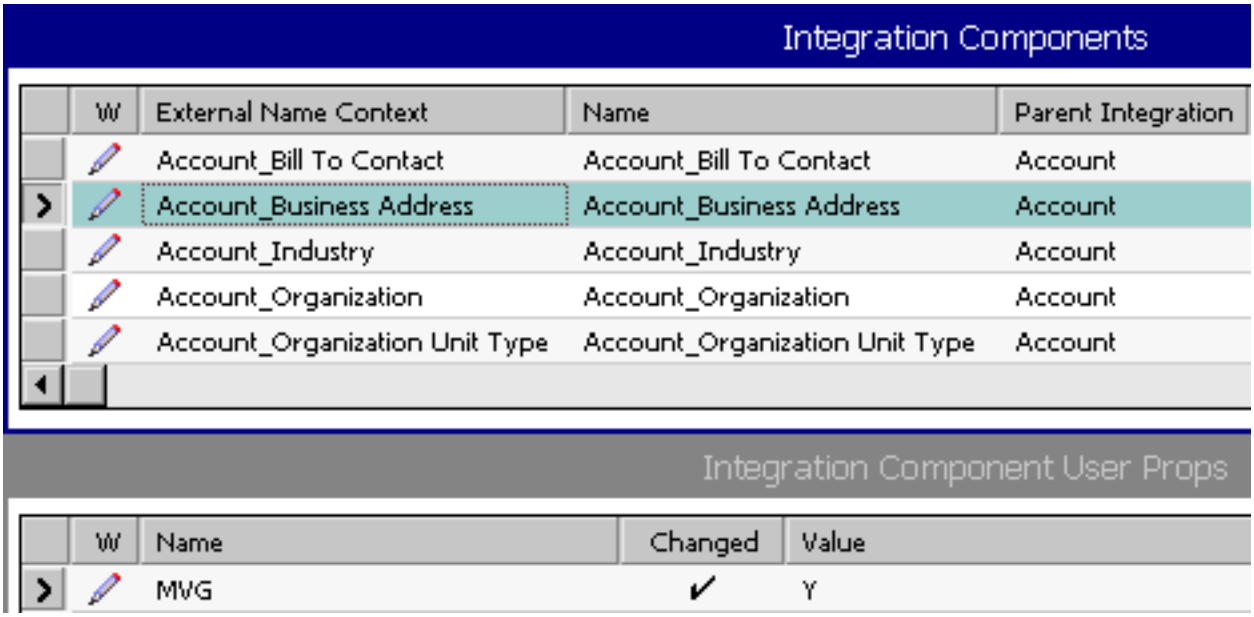
Note: Two business components are used to represent an MVG.

Creating a Siebel Integration Component to Represent an MVG

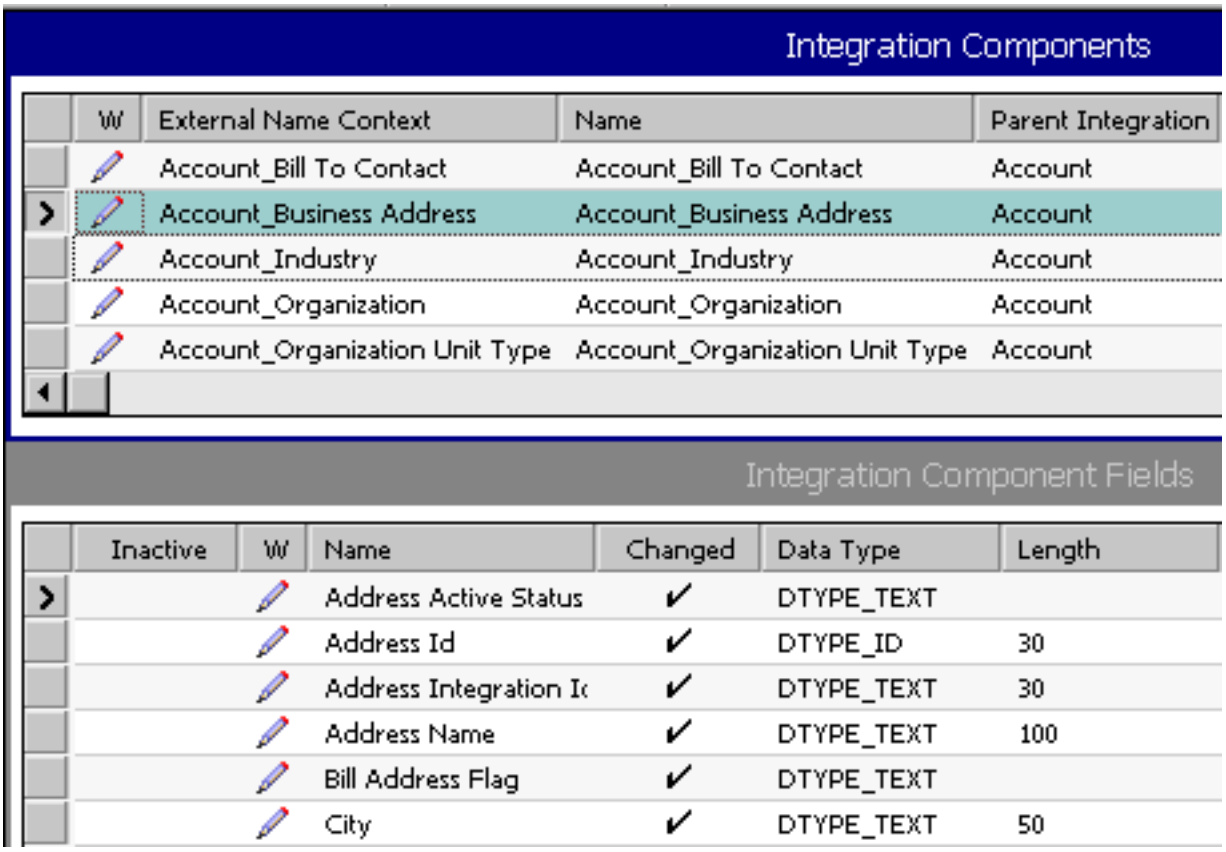
To create a Siebel integration component to represent an MVG, it is necessary also to create two integration components:

- The first integration component represents the parent business component. In the example, this is the Account business component. This integration component contains only the fields that are defined in the parent business component, but which are not based on MVGs. The Multivalue Link property and the Multivalue property are empty for these fields.
- The second integration component represents the MVG business component. In the example, this is the Business Address business component. The second integration component has one integration field for each field based on the given MVG in the parent business component. An integration component user property will be set on this integration component to tell the EAI Siebel Adapter that it is based on an MVG business component. If the MVG is a regular MVG, then the user property is named MVG. If the MVG is an Association MVG, then the user property is named MVGAssociation. In both cases, the value of the user property is Y.

The following figure shows an integration component based on an MVG and its user property value in Siebel Tools.



The EAI Siebel Adapter business service must know the names of the MVG fields as they are defined in the parent business component, which in this example is Account, and also the names of the MVG fields as they are known in the business component that represents the MVG, which in this example is Account Business Address. As shown in the following figure, the integration component fields represent the MVG.



To represent both names, each field is assigned an integration component field user property named MVGFieldName, or AssocFieldName if the integration component user property is MVGAssociation. The value of the integration component field user property is the name of the field shown in the parent business component, which in this example is Business Address.

Setting Primaries Through Multivalue Links

Primaries are set through multivalue links. However, do not use multivalue links for modifying the linked component. To modify the linked component, use links. If you must set primaries in addition to modifying the linked component, then use an MVG or MVGAssociation integration component user property set to Y, and an MVGLink integration component user property whose value is the child business component. For example, the Account_Business Address integration component of the Account IO integration object has the integration component user properties MVG (whose value is Y) and MVGLink (whose value is Business Address).

> **Note:** It is highly recommended that you use the EAI Siebel Wizard to create integration objects, so that the correct integration components and user properties will also be created. For more information, see *Creating Integration Objects Using the EAI Siebel Wizard Business Service*.

Validation of Integration Component Fields and Picklists

If an integration component field is created for a Siebel business component field, and the business component field is based on a picklist, then the EAI Siebel Adapter or the Object Manager validates the field. To have the validation done using the EAI Siebel Adapter, the integration component field has a user property with the name PICKLIST and a value of Y; otherwise, validation is done by the Object Manager.

If the EAI Siebel Adapter validates the integration component field, and if the pickmap for the picklist contains more than one field, then, when designing the integration object, you must decide the following:

- Which of the fields to use as a search criterion
- Which fields to simply update if input values are different from those in the picklist (provided that the picklist allows updates)

Note: Using the PICKLIST user property on an integration component field causes truncation to 30 characters (the length of the VAL column in the S_LST_OF_VAL Table) of the input value for searching the static picklist data.

Do not use the PICKLIST property on custom integration component fields. It is designed for static picklists, based on longer columns of the S_LST_OF_VAL table. Any input value provided in the input integration component field for search in picklist fields based on columns such as DESC_TEXT (Description field of the Picklist Generic business component) or NAME (Name field of the Picklist Generic business component) will deliver no result or an incorrect result because the string in the search expression will be truncated to 30 characters.

Example of an Integration Object Based on the Order Entry Business Object

An example is an integration object based on Order Entry business object. The root component of the Order Entry business object is Order Entry - Orders with a field Account, whose pickmap contains a large number of fields such as Account, Account Location, Account Integration Id, Currency Code, Price List, and so on. One of the tasks the integration object designer must perform is to determine which of these fields is used to identify the account for an order.

If the PicklistUserKeys user property on the integration component field that is mapped to the field with the picklist (in the previous example, Account) is not defined, then any integration component fields that are mapped to columns in the U1 index of business component's base table, and are present in the pickmap will be used by the EAI Siebel Adapter to find the matching record in the picklist (in the previous example, Account and Account Location).

In cases where the default user key for the picklist does not satisfy your business requirements (for example, you want to use only Account Integration Id instead of the default user key to pick an Account), or you want to make the user key explicit for performance reasons, then use the PicklistUserKeys user property.

The value of the PicklistUserKeys user property is a comma separated list of integration component fields that are used to find the matching record in the picklist (for example, 'Account, Account Location' or 'Account Integration Id').

For the EAI Siebel Adapter to use the fields referenced in PicklistUserKeys user property, the fields must be included in the pickmap of the underlying business component field. Note that if the business component field names and integration component field names, listed in the PicklistUserKeys property, are not the same, then the picklist must contain external names of the fields listed in the PicklistUserKeys user property.

If there is a field present in the business component and in the pickmap, and it is stored in the base table, then the EAI Siebel Adapter can use the picklist to populate this field, only if this field is present and active in the integration component. This field must also be present in the input property set, and cannot be empty.

Calculated Fields and Integration Objects

Calculated fields are inactive in an integration object when it is created. They are inactive for the following reasons:

- Performing calculations on fields requires extra processing time.
- If the calculated field is based on a field that is not included in the integration object, then errors might arise when the calculated field is updated but the field used for the calculation is not.

If your business needs require it, activate the calculated fields in integration objects.

Note: Calculated fields are those integration component fields that have the Calculated flag checked on the corresponding business component field.

Inner Joins and Integration Components

When inner joins are used, records for which the inner joined field is not set are not returned in any query. By default the wizard inactivates such fields. If your business needs require these fields, activate them.

Note: If the inner join has a join specification that is based on a required field, then the wizard does not inactivate the fields that are using that particular join.

For example, assume that Account business component has an inner join to the S_PROJ table, with Project Id field being the source field in the join specification, and the Project Name field being based on that join.

If an integration component, with an active Project Name field is mapped to the Account business component, then when this integration component is queried only accounts with Project Id field populated will be considered.

Because Project Id is not a required field in the Account business component, not every account in the Siebel Database is associated with a project. So, having Project Name active in the integration component limits the scope of the integration component to only accounts associated with a project. This typically is not desirable, so the wizard inactivates the Project Name field in this example.

If the business requirement is to include the Project Name field, but not to limit the integration component's scope to only accounts with the project, then you can change the join to S_PROJ in the Account business component to an outer join. For information on joins, see *Using Siebel Tools* .

Note: Activating an inner join can cause a query on that integration component not to find existing rows.

Defining Field Dependencies

Define dependency between fields by using the user properties of the integration component field. The names of these user properties must start with FieldDependency, and it is recommended that the value of each property contain the name of the field on which the associated field is dependent. The EAI Siebel Adapter processes fields in the order defined by these dependencies, and generates an error if cyclic dependencies exist.

The EAI Siebel Adapter automatically takes into account the dependencies of the fields set by a PickList on the fields used as constraints in that PickList. For example, if a PickList on field A also sets field B, and is constrained by field C, then this implies dependencies of both A and B on C. As a consequence, the EAI Siebel Adapter sets field C before fields A and B.

Repository Objects

For the EAI Siebel Adapter to deal with repository objects, a user property REPOBJ must be defined on the root integration component. If this property is set to Y, then the EAI Siebel Adapter sets a context on the repository so that the rest of the operations are performed in that context.

About Integration Component User Properties as Operation Controls

Each business component, link, and MVG has properties such as No Update, No Delete, and No Insert. So do integration components, in the form of integration component user properties. These user properties, listed in the following table, indicate the operations that can and cannot be performed on an object.

User Property	Description
NoDelete, NoInsert, NoQuery, NoSynchronize, NoUpdate	Indicate which operations cannot be performed on the corresponding business component. Can take the value Y or N. If any of these user properties are set to Y, then the corresponding business component method is used to validate the operation. When the business component attempts to perform a restricted operation, an error is raised. NoUpdate can also be set on integration component fields. For more information on business component properties, see *Configuring Siebel Business Applications* .
IgnorePermissionErrorsOnUpdate, IgnorePermissionErrorsOnInsert, IgnorePermissionErrorsOnDelete	Suppress the errors that arise from having the NoUpdate, NoInsert, and NoDelete user properties set to Y, respectively. The errors are ignored and processing continues.
AdminMode	When set to Y, indicates that the update of the corresponding business component is to be performed in Admin Mode. Admin Mode turns off all insert and update restrictions for the business components used by a view, including those specified by business component user properties. You can set the AdminMode user property on integration objects or integration components. For more information on Admin Mode, see *Configuring Siebel Business Applications* .

For the EAI Siebel Adapter to successfully perform an operation, that operation must be allowed at all levels. If the operation is allowed at every level except the field level, then a warning message is logged in the log file and processing continues. Otherwise, an error message is returned, and the transaction is rolled back.

Permissions on integration components are checked by the EAI Siebel Adapter, and permissions on business components, links, and fields are checked by the Object Manager.

About Integration Component Keys

There are multiple types of integration component keys:

- **User Key.** See *User Keys*.
- **Status Key.** See *Status Keys*.
- **Hierarchy Parent Key.** See *Hierarchy Parent Keys*.
- **Hierarchy Root Key.** See *Hierarchy Root Keys*.
- **Modification Key.** See *Configuring the EAI Siebel Adapter Business Service for Concurrency Control*.

Note: It is recommended to have only one integration component key for every type of key except the user key. For example, if there are two hierarchy parent keys defined for an integration component, then the EAI Siebel Adapter picks the first one and ignores the second one.

User Keys

A user key is a group of fields whose values must uniquely identify a Siebel business component record. During inbound integration, user keys are used to determine whether the incoming data updates an existing record or inserts a new one. The Integration Object Builder wizard automatically creates some user keys based on characteristics discussed in *User Key Generation Algorithm*. Make sure that the generated user keys match your business requirements; otherwise, inactivate them or add new user keys as appropriate.

In Siebel Tools, user keys are defined as Integration Component Key objects, with the Key Type property set to User Key.

Integration component keys are built by the Integration Object Builder wizard, based on values in the underlying table of the business component on which the integration component is based. Integration objects that represent Siebel business objects, and that are used in insert, update, synchronize, or execute operations, must have at least one user key defined for each integration component.

A sequence of integration component user keys is defined on each integration component definition, each of which contains a set of fields. During processing of integration component instance, the EAI Siebel Adapter chooses to use the first user key in the sequence that satisfies the condition that all the fields of that user key are present in an integration component instance. The first instance of each integration component type determines the user key used by all instances of that type.

For example, consider the Account integration object instance with only the Account Name and Account Integration Id fields present. When the EAI Siebel Adapter performs validation, it first checks the Account Name and Account Location fields (the first user key for the Account integration component). In this example, because the Account Location field is missing, the EAI Siebel Adapter moves to the second user key, Account Integration Id. The Account Integration Id field is present in the integration component instance and has a value, so the EAI Siebel Adapter uses that as the user key to match the record. Now if the same instance also had the Account Location field present, but set to null, then the EAI Siebel Adapter would pick the Account Name and Account Location combination as the user key. This is because Account Location is not a required field.

A new user key is picked for each integration object instance (root component instance). However, for the child component instances, the user key is picked based on the first child instance, and then used for matching all instances of that integration component within the parent integration component instance.

For example, if a Siebel Message contains two orders, then the user key for order items is picked twice, once for each order. Each time, the user key is selected based on the first order item record and then used for all the siblings.

Note: The EAI Siebel Adapter uses user keys to match integration component instances with business component records. Because the match is case sensitive there is a chance that records are not matched if the cases of the user key fields do not match. You can use the Force Case property on the business component field to make sure that user key fields are always stored in one case, but only if you require case-insensitive matching for performance reasons. Routine use of the Force Case property is not recommended.

Note: For performance reasons, user keys for child integration components are not included in the WHERE clause of the SQL generated to query for child component records in the Siebel database. If you must query the child component to find matching records, then consider redesigning your integration objects, such as creating a new integration object where the child component becomes the parent. For example, if Account is the parent and Asset the child, and you to query for specific assets, then create a new integration object where Asset is the parent and Account is the child.

User Key Generation Algorithm

The Integration Object Builder wizard computes the user keys by traversing several Siebel objects, including the business object, business component, table, and link. This is because not every table user key meets the requirements to be used as the basis for integration object user keys.

To understand how the Integration Object Builder wizard determines valid integration component keys, you can simulate the process of validating the user keys. For example, you can determine the table on which your business component is based by looking in Siebel Tools.

To find the user keys for a table

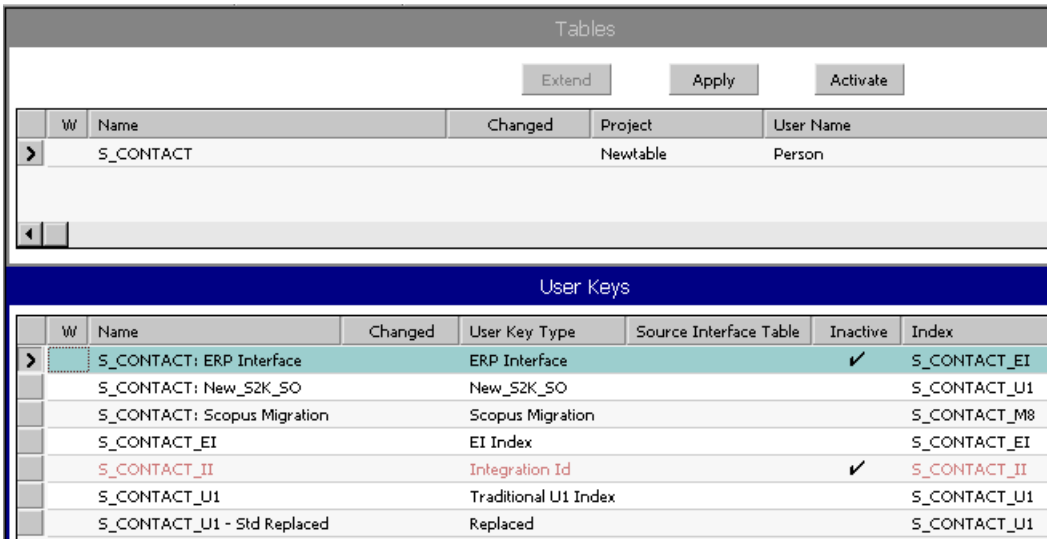
1. Select the Business Component object in the Object Explorer.

   The Business Components list appears in the Object List Editor.
2. Select a business component.
3. Click the link in the Table column.

   The Tables list appears, displaying the table associated with the business component (for example S_CONTACT).
4. Expand the Tables object in the Object Explorer, and then select User Key.

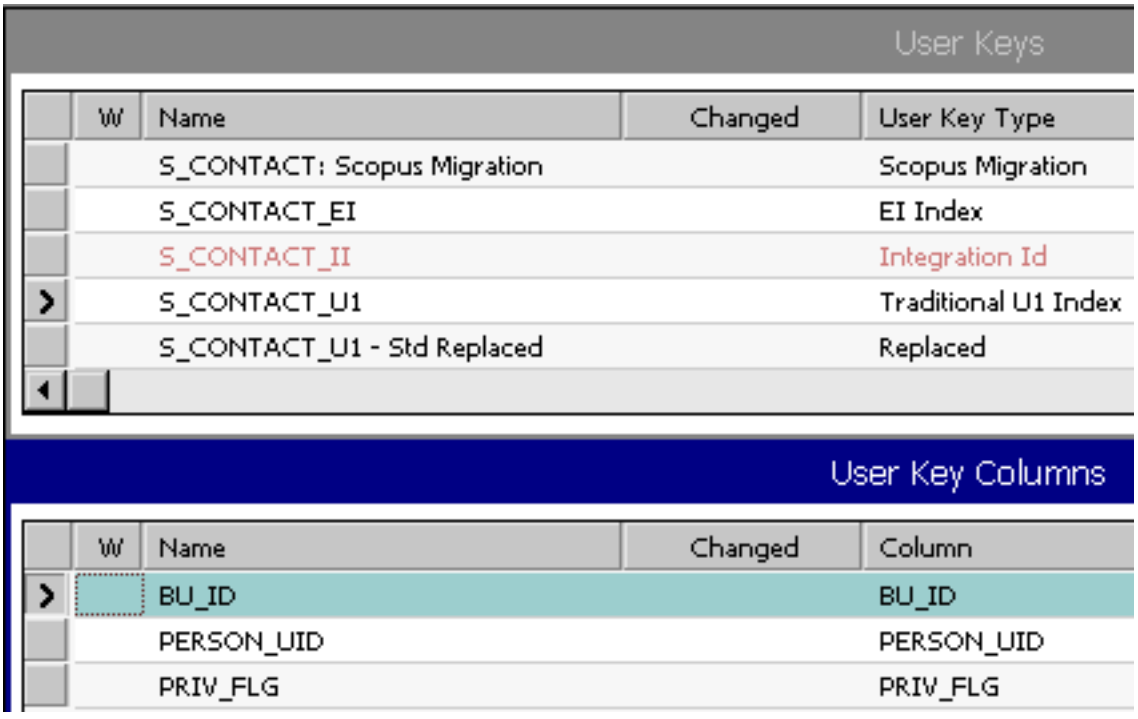
   The User Keys list displays the user keys defined for that table.

For example, as shown in the following figure. the table S_CONTACT has several user keys.



Not every user key will necessarily be valid for a given business component. Multiple business components can map to the same underlying table; therefore, it is possible that a table's user key is not valid for a particular business component, but is specific to another business component

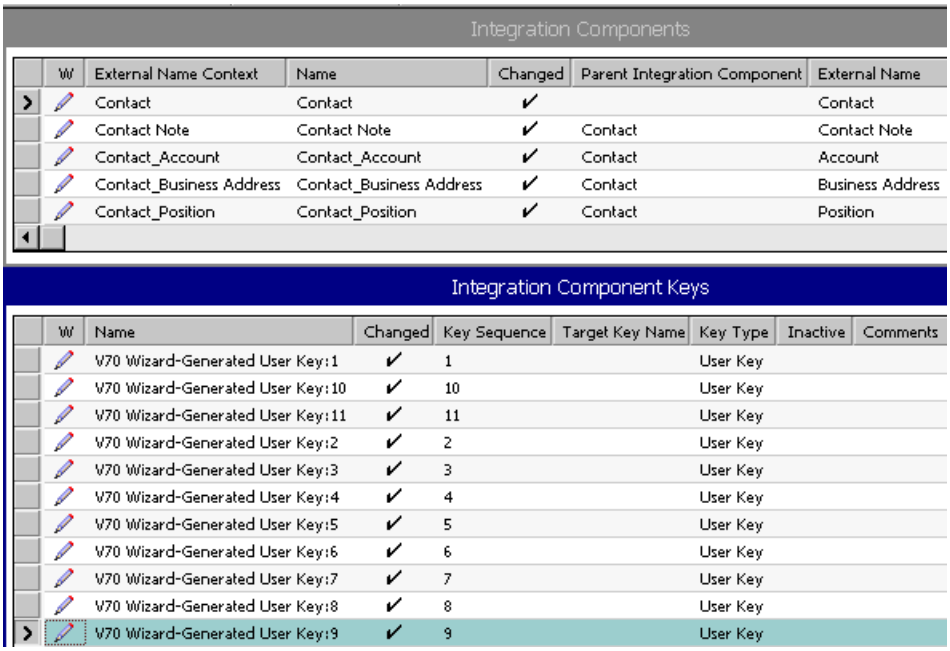
Each User Key Column child object defined for a given user key must be exposed to the business component in which you are interested. For example, the following figure shows three user key columns for the user key S_CONTACT_U1.



If the columns of the user key are exposed in the business component, and those columns are not foreign keys, then the Integration Object Builder wizard creates an integration component key based on the table's user key. The Integration

Object Builder wizard also defines one integration component key field corresponding to each of the table's user key columns.

The Integration Object Builder wizard builds the integration component keys based on these table user keys. As illustrated in the following figure, the wizard defines one integration component key for each table user key column.



Integration Components

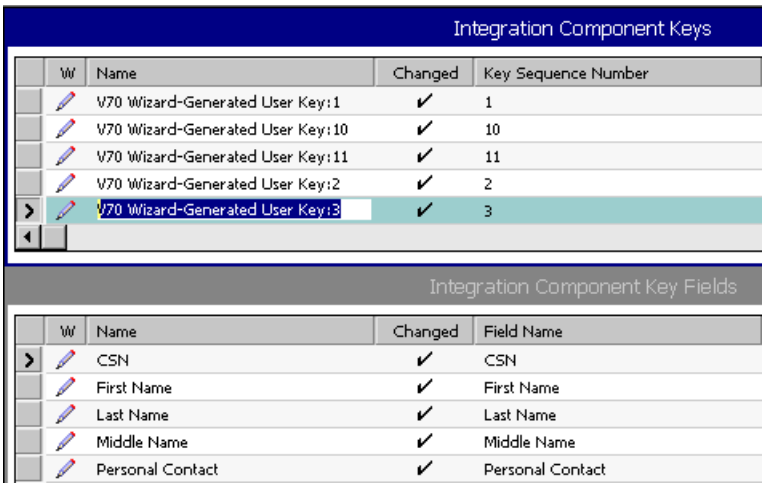
W	External Name Context	Name	Changed	Parent Integration Component	External Name
	Contact	Contact	✔		Contact
	Contact Note	Contact Note	✔	Contact	Contact Note
	Contact_Account	Contact_Account	✔	Contact	Account
	Contact_Business Address	Contact_Business Address	✔	Contact	Business Address
	Contact_Position	Contact_Position	✔	Contact	Position

Integration Component Keys

W	Name	Changed	Key Sequence	Target Key Name	Key Type	Inactive	Comments
	V70 Wizard-Generated User Key:1	✔	1		User Key		
	V70 Wizard-Generated User Key:10	✔	10		User Key		
	V70 Wizard-Generated User Key:11	✔	11		User Key		
	V70 Wizard-Generated User Key:2	✔	2		User Key		
	V70 Wizard-Generated User Key:3	✔	3		User Key		
	V70 Wizard-Generated User Key:4	✔	4		User Key		
	V70 Wizard-Generated User Key:5	✔	5		User Key		
	V70 Wizard-Generated User Key:6	✔	6		User Key		
	V70 Wizard-Generated User Key:7	✔	7		User Key		
	V70 Wizard-Generated User Key:8	✔	8		User Key		
	V70 Wizard-Generated User Key:9	✔	9		User Key		

Each valid integration component key contains fields. For example, as shown in the following figure, for the Contact integration component, User Key 3 is made up of five fields: CSN, First Name, Last Name, Middle Name, and Personal Contact.

CAUTION: Only modify user keys if you have a good understanding of the business component and integration logic.



Integration Component Keys

W	Name	Changed	Key Sequence Number
	V70 Wizard-Generated User Key:1	✔	1
	V70 Wizard-Generated User Key:10	✔	10
	V70 Wizard-Generated User Key:11	✔	11
	V70 Wizard-Generated User Key:2	✔	2
	V70 Wizard-Generated User Key:3	✔	3

Integration Component Key Fields

W	Name	Changed	Field Name
	CSN	✔	CSN
	First Name	✔	First Name
	Last Name	✔	Last Name
	Middle Name	✔	Middle Name
	Personal Contact	✔	Personal Contact

When the Integration Object Builder wizard creates these integration component keys, it attempts to use the appropriate table user keys, that is the user keys that help to uniquely identify a given record. In some cases, you might find that certain integration component keys created by the Integration Object Builder wizard are not useful for your particular needs. In that case, you can manually inactivate the keys you do not want to use by checking the Inactive flag on that particular user key in Siebel Tools. You can also inactivate user key fields within a given user key.

> **Note:** For ease of maintenance and upgrade, inactivate unnecessary generated user keys and user key fields instead of deleting them.

Status Keys

It is useful to know the status of your integrations. For example, if you are sending an order request, then you might want to know the ID of the Order created so that you can query on the order in the future. You can set the StatusObject method argument of the EAI Siebel Adapter business service to true to return an integration object instance as a status object.

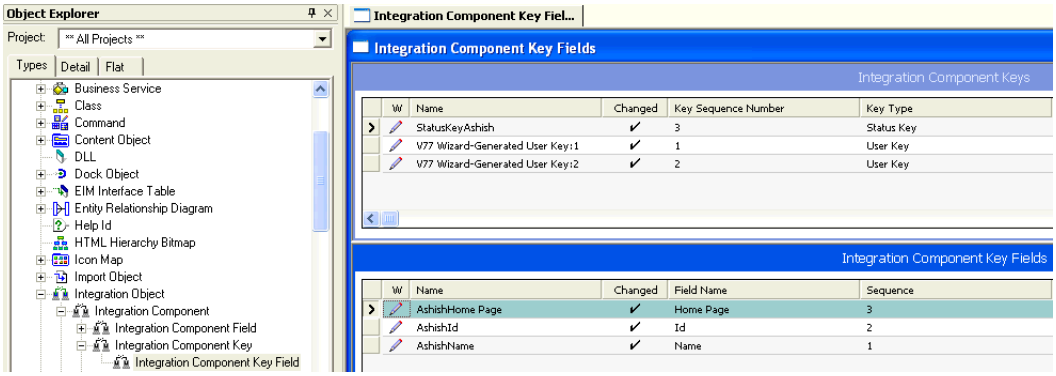
The status returned is defined in the Integration Component using Status Keys. A Status Key is an Integration Component key of the type Status Key. Fields defined as part of the Status Key are included in the returned StatusObject.

If a Status Key is not defined for the Integration Component then neither the component nor any of its children are included in the returned object:

- To include descendants of an Integration Component without including any of its fields in the returned status object, specify an empty Status Key.
- To include information about which one of the update, insert, or delete operations was performed during an upsert request or synchronize request, include a field named *Operation* in the Status Key.

Status Key Examples

For example, the AccountAshish integration object has an Account integration component with an integration component key called StatusKeyAshish, with the integration component key fields AshishName, AshishId, and AshishHomePage (the following figure).



Example with No Status Object

When no StatusObject business service method argument is defined, as in this input XML file for an upsert operation using the EAI Siebel Adapter business service

```
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet>
<SiebelMessage MessageId="42-1PGR" IntObjectName="AccountAshish"
MessageType="Integration Object" IntObjectFormat="Siebel Hierarchical">
 <ListOfAccountAshish>
<Account Main_spcPhone_spcNumber="+33123456789" Primary_spcOrganization="Default
Organization" Home_spcPage="mycompany.com" Location="France" Name="Ashish 9
Telecom"/>
 </ListOfAccountAshish>
 </SiebelMessage>
</PropertySet>
```

all of the fields in the integration component are returned:

```
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet ErrorContextIntComp="" ErrorContextSearchSpec="" OMErrorCode=""
PrimaryRowId="42-C739Q" OMErrorSymbol="" ErrorCode="0x0" ErrorSymbol="">
<SiebelMessage MessageId="42-1PGR" MessageType="Integration Object"
IntObjectName="AccountAshish" IntObjectFormat="Siebel Hierarchical">
 <ListOfAccountAshish>
 <Account Main_spcPhone_spcNumber="+33123456789"
 Primary_spcOrganization="Default Organization" Home_spcPage="mycompany.com"
 Location="France" Name="Ashish 9 Telecom"/>
 </ListOfAccountAshish>
</SiebelMessage>
</PropertySet>
```

Example with Status Object

When the StatusObject method argument is set to true, as in this input XML file:

```
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet StatusObject="true">
<SiebelMessage MessageId="42-1PGR" IntObjectName="AccountAshish"
MessageType="Integration Object" IntObjectFormat="Siebel Hierarchical">
 <ListOfAccountAshish>
 <Account Main_spcPhone_spcNumber="+33123456789"
 Primary_spcOrganization="Default Organization" Home_spcPage="mycompany.com"
 Location="France" Name="Ashish 9 Telecom"/>
 </ListOfAccountAshish>
</SiebelMessage>
</PropertySet>
```

only the fields in the status key are returned:

```
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet ErrorContextIntComp="" ErrorContextSearchSpec="" OMErrorCode=""
PrimaryRowId="42-C739Q" OMErrorSymbol="" ErrorCode="0x0" ErrorSymbol="">
<SiebelMessage MessageId="42-1PGR" MessageType="Integration Object"
IntObjectName="AccountAshish" IntObjectFormat="Siebel Hierarchical">
 <ListOfAccountAshish>
 <Account Id="42-C739Q" Home_spcPage="mycompany.com" Name="Ashish 9 Telecom"/>
 </ListOfAccountAshish>
 </SiebelMessage>
</PropertySet>
```

For more information on the EAI Siebel Adapter business service, see *EAI Siebel Adapter Business Service*.

Hierarchy Parent Keys

The Hierarchy Parent Key is used for integration objects that have a homogeneous hierarchy. This key must have only the Parent Id. The Hierarchy Parent Key is used for maintaining the hierarchy and keeping the data normalized.

For example, when you insert quotes, each quote item in turn can have more quote items. In this case, the first quote item inserted by the EAI Siebel Adapter has the Parent Id set to blank, but for each child quote item, the EAI Siebel Adapter checks the keys to figure out which fields are to be set. If the Hierarchy Parent Key is not defined, then the child quote item is inserted as a new quote item without a link to its parent (denormalized).

Note: You cannot rearrange the hierarchy after it has been created. For example, if A is a parent of B, and you try to upsert B as a parent of A, then an error will occur. Instead you must delete the hierarchy and then re-create it.

Hierarchy Root Keys

The Hierarchy Root Key is an optional key that is useful only when integration objects have a homogeneous hierarchy. You can use this key to improve performance. The Hierarchy Root Key must have only one field, Root Id, which the EAI Siebel Adapter populates with the value of the ID field in the component instance that is in the root of the homogenous hierarchy. For example, assume quote Q1 has quote items A, B, and C where each of the quote items has child quote items (A1, A2, B1, B2, ...). If you want to update the quantity requested for all quote items starting with the root quote item B, then it is faster if the data is denormalized. Using the Hierarchy Root Key, you can search for all records with Root Id equal to the Row Id of B, and set the QuantityRequested field for each item.

Note: When the business component is hierarchy enabled, then the wizard automatically sets the Hierarchy Parent Key for the complex integration component. To have a business component hierarchy enabled you must set the property Hierarchy Parent Field.

About EAI Siebel Adapter Access Control

You can use the following mechanisms to control the access of the EAI Siebel Adapter to the database:

- **Restricted access to a static set of integration objects.** You can configure the EAI Siebel Adapter business service, or any business service that is based on the CSEEAISiebelAdapterService class, to restrict access to a static set of integration objects. To do this, set a business service user property called AllowedIntObjects, which contains a comma-separated list of integration object names that this configuration of the EAI Siebel Adapter can use. This allows you to minimize the number of integration objects your users must expose outside of Siebel CRM through HTTP inbound or MQSeries Receiver server components. If this user property is not specified, then the EAI Siebel Adapter uses any integration objects defined in the current Siebel Repository.
- **ViewMode.** You can specify the visibility mode of business components that the EAI Siebel Adapter uses. This mode is specified as the integration object user property ViewMode. This user property can take different values, as defined by LOV type REPOSITORY_BC_VIEWMODE_TYPE.

Note: For information on ViewMode, see *Siebel Tools Online Help* .

3 Creating and Maintaining Integration Objects

Creating and Maintaining Integration Objects

This chapter describes how to use the Integration Object Builder wizard in Siebel Tools to create new Siebel integration objects. This wizard guides you through the process of selecting objects (either from the Siebel repository or from an external system) on which you can base your new Siebel integration object. This chapter also describes how to fine-tune and refine the integration object you have created.

The chapter consists of the following topics:

- *About the Integration Object Builder*
- *About the EAI Siebel Wizard Business Service*
- *Process of Creating Integration Objects*
- *Creating Integration Objects Using the EAI Siebel Wizard Business Service*
- *Creating an Integration Object Based on Another Root Business Component*
- *Creating an Integration Object with Many-To-Many Relationships*
- *Creating Integration Object Instances Programmatically*
- *Guidelines for Configuring Integration Objects*
- *Validating Integration Objects*
- *Testing Integration Objects*
- *Deploying Integration Objects to the Run-Time Database*
- *About Synchronizing Integration Objects*
- *Synchronizing Integration Objects*
- *Resolving Synchronization Conflicts for Integration Objects and User Properties*
- *Using Formatted Values in Integration Objects*
- *Generating Integration Object Schemas*
- *Optimizing the Performance of Integration Objects*
- *About Business Component Restrictions for Integration Components*
- *Guidelines for Using Integration Components*

About the Integration Object Builder

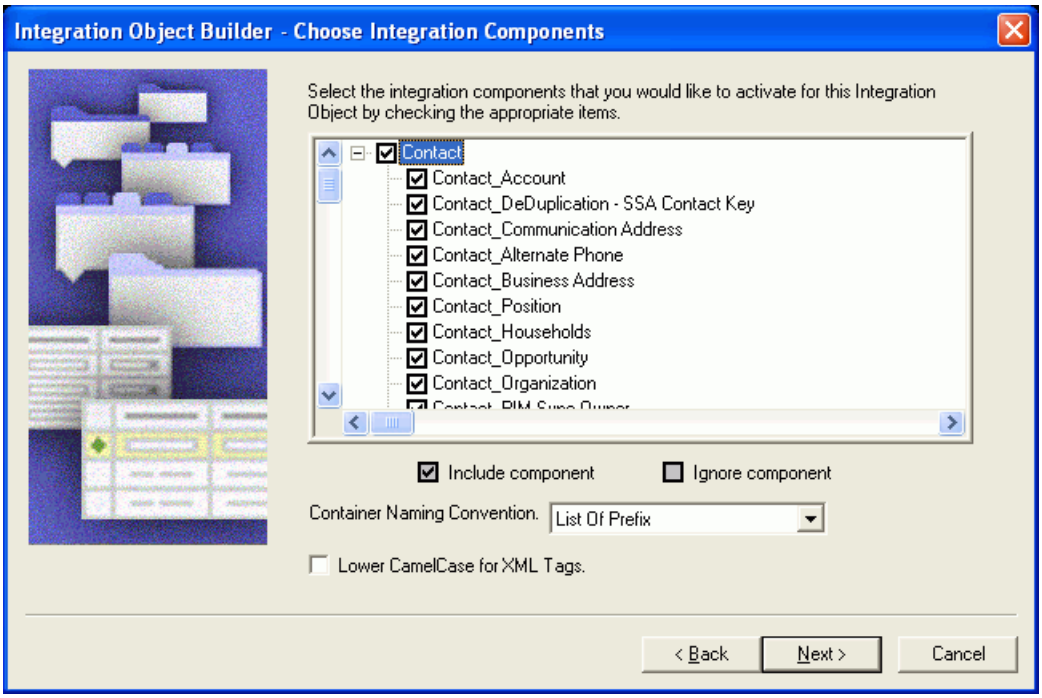
The Integration Object Builder wizard in Siebel Tools builds a list of valid components from which you can choose the components to include in your Siebel integration object.

Note: The Integration Object Builder provides a partial rendering of your data in the integration object format. You must review the integration object definition and complete the definition of your requirements. In particular, confirm that the user key definitions are defined properly. You might have to enter keys and user properties manually or inactivate unused keys and fields in Siebel Tools. Do not expect to use the integration object without modification.

About the EAI Siebel Wizard Business Service

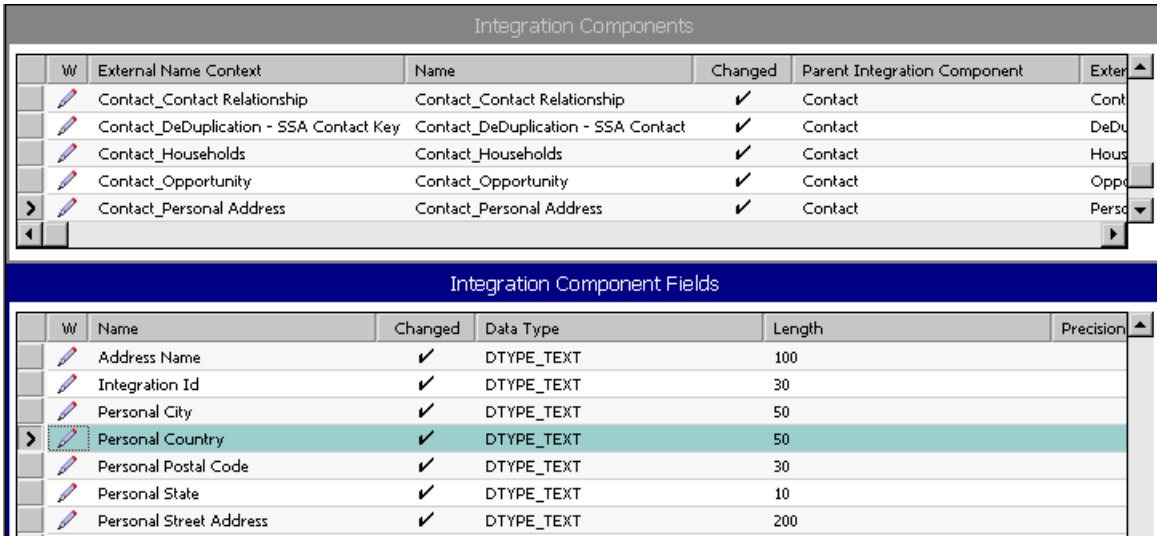
You can use the Integration Object Builder to create integration objects that represent Siebel business objects. During the process of creating a new integration object, described in *Creating Integration Objects Using the EAI Siebel Wizard Business Service*, you can choose the EAI Siebel Wizard as the business service to help create the object. This wizard understands the structure of Siebel business objects. It returns a list of the available business objects on which you can choose to base your integration object.

The wizard also returns a list of the available components contained within the object you have chosen, shown in the following figure. When you select certain components in the wizard, you are activating those components in your integration object. Your integration object contains the entire structural definition of the business object you selected in the first wizard dialog box. Only the components you checked, or left selected, are active within your integration object. That means any instances you retrieve of that integration object contains only data represented by the selected components.



After the wizard creates your integration object, you can edit the object in Siebel Tools, as shown in the following figure. You might choose to drill down into the integration components and activate or inactivate particular components or even particular fields within one or more components.

Note: Always inactivate the fields rather than delete them. When you execute the synchronization task, using the Integration Object Synchronize wizard in Siebel Tools, inactivated fields remain inactive, while the deleted fields are created as active fields in the integration object.



Process of Creating Integration Objects

Perform the following tasks to create an integration object:

1. Log in to Siebel Tools or Web Tools as an administrator (see *Using Siebel Tools*).
2. Create a workspace.
3. *Creating Integration Objects Using the EAI Siebel Wizard Business Service*
4. (Optional) Configuring the integration object (see *Guidelines for Configuring Integration Objects*)
5. *Validating Integration Objects*
6. Deliver the changes to the Integration Branch.
7. *Testing Integration Objects*
8. (Optional) *Deploying Integration Objects to the Run-Time Database*

Creating Integration Objects Using the EAI Siebel Wizard Business Service

Siebel Tools provides the EAI Siebel Wizard business service to walk you through creating an integration object. Use this wizard to create your integration object.

You can also use the wizard to deploy integration objects to the run-time database.

> **Note:** If you deploy integration objects while the Siebel Server is running, then you must subsequently clear the Web services cache in the Administration - Web Services screen, Inbound (or Outbound) Web Services view.

This task is a step in *Process of Creating Integration Objects*.

To create a new Siebel integration object

1. In Siebel Tools, create a new project and lock it, or lock an existing project in which you want to create your integration object.
2. From the File menu, choose New Object to display the New Object Wizards dialog box.
3. Select the EAI tab, and then double-click Integration Object.

   The Integration Object Builder wizard appears.
4. On the first page of the Integration Object Builder wizard:

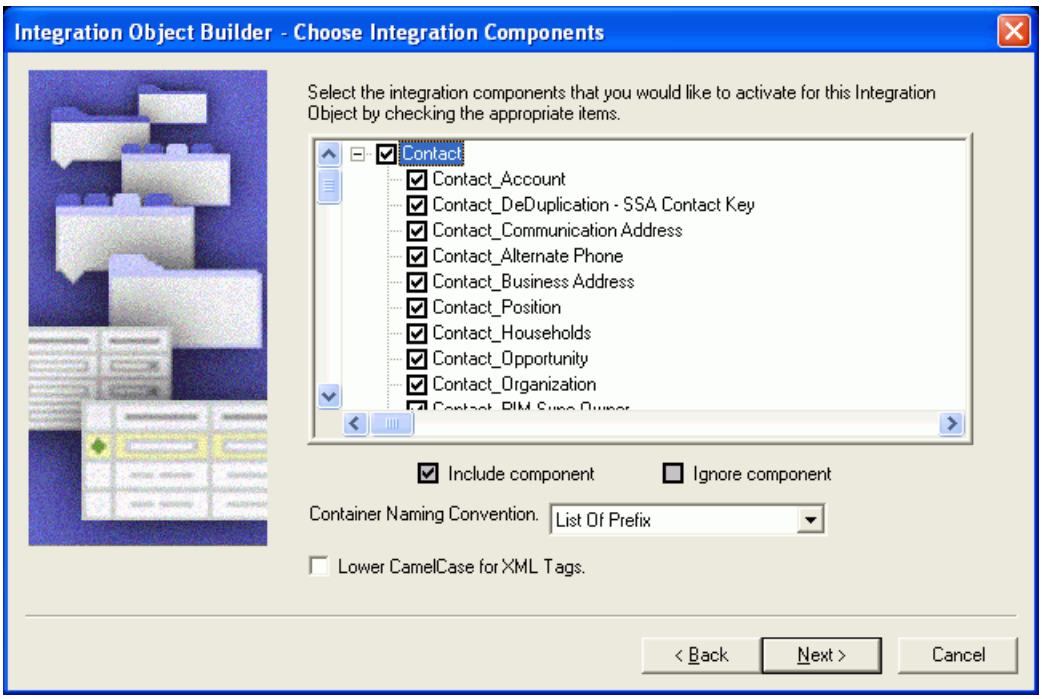
   a. Select the project you locked in Step 1.
   b. For the source system, select the EAI Siebel Wizard business service.
5. Click Next.

   The second page of the Integration Object Builder wizard appears.

   a. Select the source object (business object). This is the object model for the new Siebel integration object. Only business objects with Primary Business Components appear on this picklist.
   b. Select the source root (business object component).
   c. Type a unique name in the field for the new Siebel integration object and click Next.

   > **Note:** The name of an integration object must be unique among other integration objects. There will be an error if the name already exists.

   The next page of the wizard, the Integration Object Builder - Choose Integration Components page, displays the available components of the object you chose.



6. Deselect the components you want the wizard to ignore. This means you cannot integrate data for that component between the Siebel application and another system.

> **Note:** Any component that has a plus sign (+) next to it is a parent in a parent-child relationship with one or more child components. If you deselect the parent component, then the children following that component are deselected as well. You cannot include a child component without also including the parent. The Integration Object Builder enforces this rule by automatically selecting the parent of any child you choose to include.

For example, assume you have chosen to build your Siebel integration object on the Siebel Account business object, and you want to create an integration component based on the Account and Contact business components:

a. Deselect the Account integration component at the start of the scrolling list. This action deselects the entire tree after Account.
b. Select the Contact component. When selecting a child component, its parent component is also selected, but none of the components after the child component are selected. You must individually select the ones you want.

7. From the Container Naming Convention drop-down menu, choose either `List of Prefix` or `Suffix s`.

   This allows flexibility when generating XML Schema Definition (XSD) files from integration objects. For example, rather than generating container elements such as xsd:ListOfContact, you can choose to have elements generated named xsd:Contacts.
8. Select the Lower CamelCase for XML Tags check box to use this convention for naming XML tags.

   CamelCase is a naming convention in which a name is formed of multiple words that are joined together as a single word, with the first letter of each of the multiple words capitalized so that each word that makes up the name can easily be read. The name derives from the hump or humps that seem to appear in any Camel Case name. In Lower CamelCase, the first letter of the name is lowercase, for example `myNewIntegrationObject`.

   The default convention is Upper CamelCase, for example MyNewIntegrationObject.
9. Click Next. The next page displays error or warning messages generated during the process. Review the messages, and take the appropriate actions to address them.
10. (Optional) Select the Deploy the Integration Object check box to deploy the integration object to the run-time database.

    For more information, see *Deploying Integration Objects to the Run-Time Database*.
11. Click Finish.

    Your new Siebel integration object appears in the list of integration objects in Siebel Tools.

    On the Integration Components screen, the Account integration component is the only component that has a blank field in the Parent Integration Component column. The blank field identifies Account as the root component. The Siebel integration object also contains the other components selected, such as Contact and its child components.

> **Note:** When you create your integration object based on a Siebel business object, do not change its integration component's External Name Context; otherwise, the synchronization process will not recognize the integration component, and will remove it from the integration object.

12. To view the fields that make up each integration component, select a component from the integration component list in Siebel Tools.

    The Integration Component Fields list displays the list of fields for that component. Note the system fields Conflict Id, Created, Id, Mod Id, Updated, operation, and searchspec in the list. This setting prevents the EAI

Siebel Adapter Query and QueryPage method from outputting these fields. For more details, see *About Using Language-Independent Code with the EAI Siebel Adapter Business Service*.

13. When finished, compile the locked project.

Creating an Integration Object Based on Another Root Business Component

The Integration Object Builder wizard, using the EAI Siebel Wizard, allows you to choose which business object to use. However, the Integration Object Builder wizard will generate the Primary Business Component as the root Integration Component. If it happens that the business object contains multiple root business components (note the difference between root and primary business component), and that the user requires the Integration Object to be created based on another root business component, then you perform the following procedure.

To create an integration object based on another root business component

1. In Siebel Tools, lock the project containing the business object you want to modify.
2. Modify the business object definition to have that particular root business component as the Primary Business Component.
3. Run the Integration Object Builder wizard and choose the business object you want to use.
4. Undo the changes to the business object definition that you made in Step 2.

   > **Note:** This is necessary because unless you are certain about what you are doing in terms of changing the Primary Business Component of the business object, it is recommended that you roll back the changes so that they do not affect any business logic.

5. Compile the locked project.

Creating an Integration Object with Many-To-Many Relationships

The following is an example of how to create an integration object with two components that have a many-to-many (M:M) relationship. In this example, an integration object uses the Contact business object and the Contact and Opportunity business components.

To create an integration object with a many-to-many business component

1. In Siebel Tools, create a new project and lock it, or lock an existing project in which you want to create your integration object.
2. From the File menu, choose New Object to display the New Object Wizards dialog box.
3. Select the EAI tab, and then double-click the Integration Object icon.
4. In the Integration Object Builder wizard:
   a. Select the project you locked in Step 1.
   b. Select the EAI Siebel Wizard business service.

5. Click Next and in the second page of the Integration Object Builder wizard:
    a. Select the source object Contact to be the base for the new Siebel integration object.
    b. Type a unique name in the field for the new Siebel integration object, for example Sample Contact M:M, and then click Next.
    c. Select the source root for the new integration object from the list.
6. From the list of components, select Contact and Opportunity.

    Note: There is also a component named Contact_Opportunity in the list. This component is an MVGAssociation component, and you pick it only if you need this integration object to set the primary opportunity for contact. For information on multivalue groups, see *About the Search Spec Input Method Argument*.

7. Inactivate all integration component fields in the Contact integration component except First Name, Last Name, Login Name, and Comment. (In this example, these are the only fields you need for Contact.)
8. Inactivate all integration component fields in the Opportunity integration component except Account, Account Location, Budget Amt, Name, and Description. (In this example, these are the only fields you need for Opportunity.)
9. Compile the locked project.

Creating Integration Object Instances Programmatically

Because integration objects adhere to a set of structural conventions, they can be traversed and transformed programmatically, using Siebel eScript objects, methods, and functions, or transformed declaratively using the Siebel Data Mapper.

This topic outlines the steps required to create an integration object instance programmatically, using the EAI Account integration object as an example.

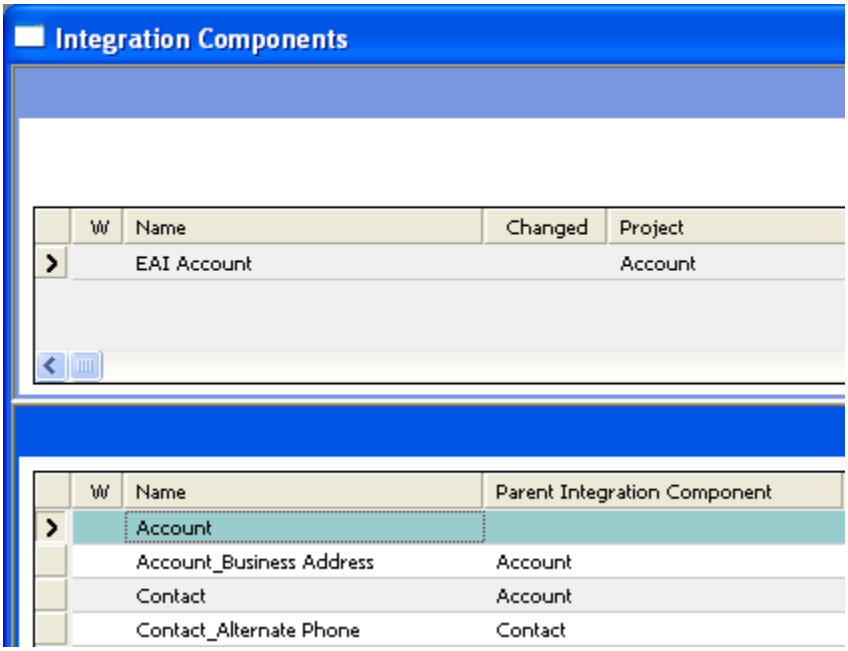
To create the correct integration object instance programmatically, follow these rules:

- The root property set must have its type set to ListOf concatenated with the integration object name (ListOf*IOName*).
- The next property set of the hierarchy must have the root integration component name as its type. The root integration component is the one that has no Parent Integration Component set (*RootICName*).
- All other integration components must have the Parent Integration Component set. For those integration components, create a property set with type set to ListOf concatenated with the integration component name (ListOf*ICName*) and then add as child to this property set another one with type set to the integration component name.

The following hierarchy demonstrates the rules:

```
ListOfIOName
RootICName
ListOfICName1
ICName1
ListOfICName1_1
ICName1_1
ListOfICName2
ICName2
```

The following figure shows some of the integration components in the hierarchy of the EAI Account integration object.



Based on its hierarchy, the integration object instance will have the following property set hierarchy:

```
ListOfEAI Account
Account
ListOfAccount_Business Address
Account_Business Address
ListOfContact
Contact
ListOfContact_Alternate Phone
Contact_Alternate Phone
```

The following Siebel eScript example creates an instance of the hierarchy shown in the previous figure:

```
// Local variable creation, error handling, and object destruction are omitted for
clarity.
psConAltPhone.SetType("Contact_Alternate Phone");
psConAltPhone.SetProperty("Alternate Phone #", "555-5555");
psListOfConAltPhone.SetType("ListOfContact_Alternate Phone");
psListOfConAltPhone.AddChild(psConAltPhone);

psContact.SetType("Contact");
psContact.SetProperty("First Name", "John");
psContact.SetProperty("Last Name", "Smith");
psContact.AddChild(psListOfConAltPhone);

psListOfContact.SetType("ListOfContact");
psListOfContact.AddChild(psContact);

psAccBusAdd.SetType("Account_Business Address");
psAccBusAdd.SetProperty("Email Address", "john.smith@email.com");

psListOfAccBusAdd.SetType("ListOfAccount_Business Address");
psListOfAccBusAdd.AddChild(psAccBusAdd);

psAccount.SetType("Account");
psAccount.SetProperty("Name", "MyAccount");

// Add the children to the Account IC.
psAccount.AddChild(psListOfAccBusAdd);
psAccount.AddChild(psListOfContact);
```

```
psListOfEAIAccount.SetType("ListOfEAI Account");
psListOfEAIAccount.AddChild(psAccount);

...
```

Guidelines for Configuring Integration Objects

After you create your integration object you can configure it based on your business requirements. The following is a list of guidelines for configuring an integration object:

- In Siebel Tools, inactivate the fields that do not apply to your business requirements.
- If necessary, activate the fields that have been inactivated by the Siebel Wizard. For information, see *Integration Objects*.
- Add the fields that have not been included by the Siebel Wizard, including custom integration component fields. For information on the implications of adding or activating such fields, see *Custom Integration Component Fields*, *Calculated Fields and Integration Objects*, and *Inner Joins and Integration Components*.
- Validate the user keys. For information, see *Integration Objects*.
- Update the user properties for your integration object to reflect your business requirements. For information, see:
  - *Resolving Synchronization Conflicts for Integration Objects and User Properties*
  - *Using Formatted Values in Integration Objects*

This task is a step in *Process of Creating Integration Objects*.

Validating Integration Objects

When you have created your integration object and made the necessary modifications to meet your business requirements, you must validate it.

This task is a step in *Process of Creating Integration Objects*.

To validate your integration object

1. In Siebel Tools, select your integration object.
2. Right-click the integration object and select Validate.
3. Review the report, and modify your integration object as needed.

Note: Before creating or modifying any integration object, you need to create and open a workspace. After validation, the integration objects you create in Siebel Tools must be delivered.

Testing Integration Objects

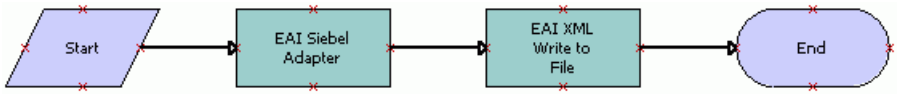
After validating and compiling integration objects, test them using the Workflow Simulator. For information on creating, modifying, and simulating workflows, see *Siebel Business Process Framework: Workflow Guide* .

This task is a step in *Process of Creating Integration Objects*.

To test a newly created integration object

1. In Siebel Tools, select the Workflow Process object in the Object Explorer.
2. Create a new workflow that runs the EAI Siebel Adapter business service against the new integration object.

   For example, create a workflow to query with the new integration object and write the output message to an XML file, as in the following:



3. Test the workflow using the Workflow Simulator.

Deploying Integration Objects to the Run-Time Database

You can deploy integration objects, which you have created in Siebel Tools, to the Siebel run-time database. Siebel object manager processes build a cache of the deployed integration objects to improve performance. These deployed integration objects are read from the cache at run time.

This saves time by allowing you to modify integration object definitions without having to shut down your production environment, create and open a workspace in Siebel Tools or Web Tools, edit integration objects in Siebel Tools, and deliver the changes to the Integration Branch.

Integration objects are read first from the cache and then from the runtime repository. The deployed integration objects are maintained in the object manager cache so that performance is not slowed by rereading these integration objects from the run-time database.

This task is a step in *Process of Creating Integration Objects*.

Deploying an Integration Object to the Run-Time Database

The following procedure is used to deploy integration objects that have already been created in Siebel Tools. To deploy an integration object while creating it with the Integration Object Builder wizard, see *Creating Integration Objects Using the EAI Siebel Wizard Business Service*.

If you make changes in Siebel Tools to a deployed integration object, then you must redeploy it. If you do not redeploy it, then the object definitions will differ between Siebel Tools and your production environment, which can cause unexpected application behavior.

To deploy an integration object to the run-time database

1. In the Object Explorer in Siebel Tools, select Integration Object.

   The Integration Objects list appears.
2. Right-click the integration object to deploy, and then choose Deploy to Runtime Database.

   The integration object is deployed.
3. In the Siebel client, navigate to the Administration- Web Services screen, Inbound (or Outbound) Web Services view.
4. Click Clear Cache to invalidate the integration object and Web services definitions in the run-time database.

   Note: Object definitions are reloaded when requested in the client.

Deployed integration objects are shown in the Administration - Web Services screen, Deployed Integration Objects view in the Siebel client.

Removing an Integration Object from the Run-Time Database

You can also remove deployed integration objects.

To remove a deployed integration object from the run-time database

1. In the Object Explorer in Siebel Tools, select Integration Object.

   The Integration Objects list appears.
2. Right-click the integration object to remove, and then choose Undeploy.

   The integration object is removed from the run-time database.
3. In the Siebel client, navigate to the Administration- Web Services screen, Inbound (or Outbound) Web Services view.
4. Click Clear Cache to invalidate the integration object and Web services definitions in the run-time database.

   Note: Object definitions are reloaded when requested in the client.

About Synchronizing Integration Objects

Business objects often require updates to their definitions to account for changes in data type, length, edit format, or other properties. It is common to want to alter database metadata, but if you do so you have to also update your integration objects to account for these updates. Otherwise, you can cause undesirable effects on your integration projects.

Some examples of these changes are:

- A field removed
- A new required field
- A new picklist for a field
- A change of relationship from one-to-many to many-to-many
- An upgrade to a new version of Siebel CRM

To help simplify the synchronization task, Siebel EAI provides the Integration Object Synchronize wizard. Although the process of synchronizing your integration object with its underlying business object is straightforward, review the integration objects you have modified to make sure that you have not inadvertently altered them by performing a synchronization. After synchronization, validate your integration object.

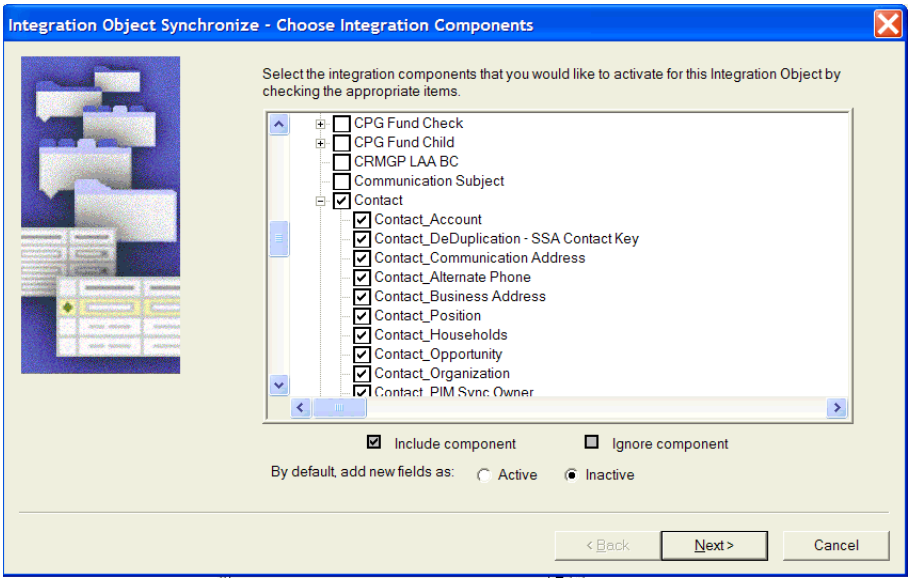
> **Note:** If business object changes are minor, such as adding a new single-value field, then it is best to synchronize the integration object. However, if business object changes are extensive, such as creating a new multivalue group, then it might be better to delete and re-create the integration object.

The following topics are also covered:

- *Synchronization Rules*
- *Updating the Entire Integration Object*
- *Deleting a Component from the Integration Object*
- *Guidelines for Maintaining Integration Objects*

Synchronization Rules

During the synchronization process, the wizard follows particular update rules. Consider a simple example involving the Siebel Account integration object with only Contact and its child components marked as active in the object. The following figure helps you to visualize this example.

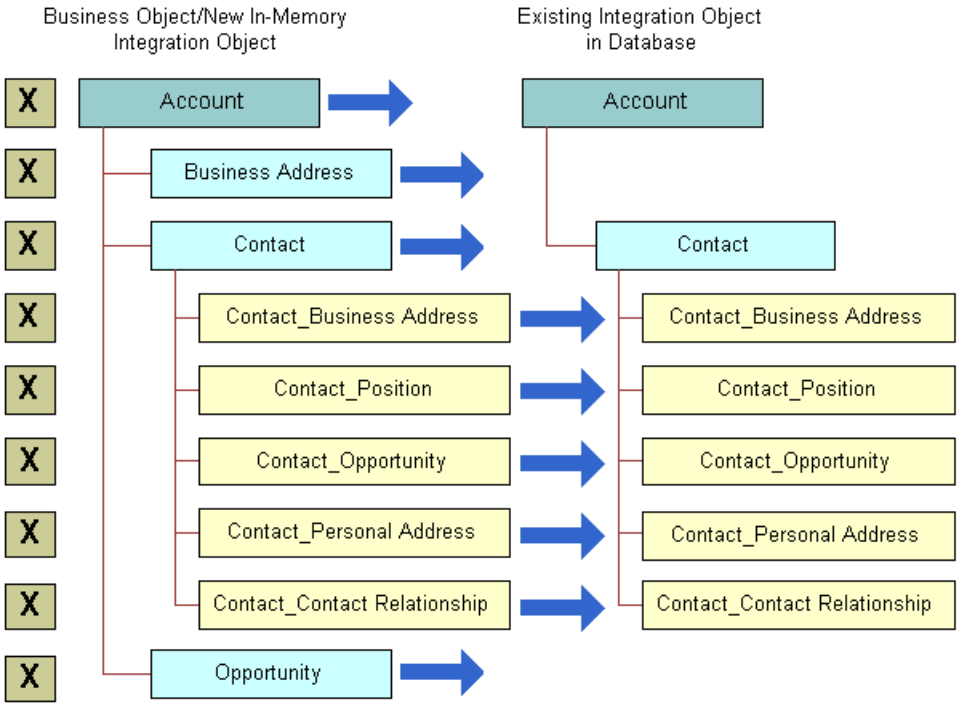


Because the Account component is the parent of Contact, it is also selected, even though you cannot see it in the previous figure.

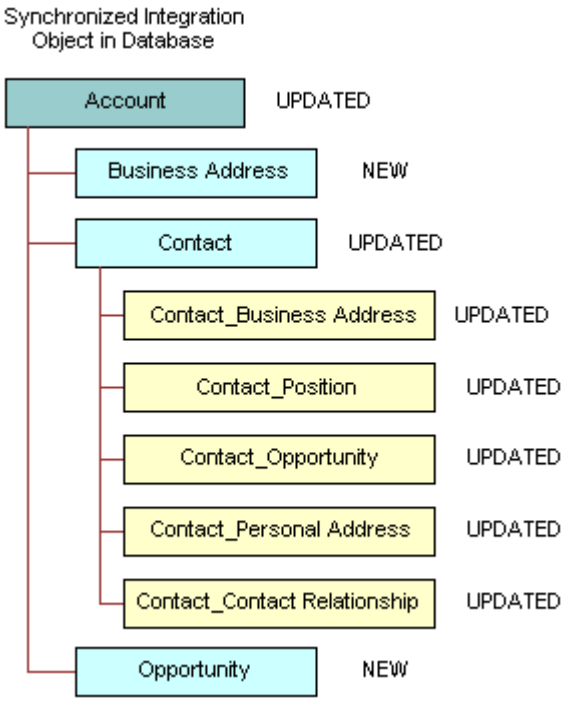
Updating the Entire Integration Object

Either the business object or the integration object might have changed since the integration object was first created. The Synchronization wizard creates a new object that takes into account any business object and integration object changes.

The following figure illustrates how the Synchronization wizard takes into account any changes.



The following figure shows how the resulting integration object is structured after the synchronization.

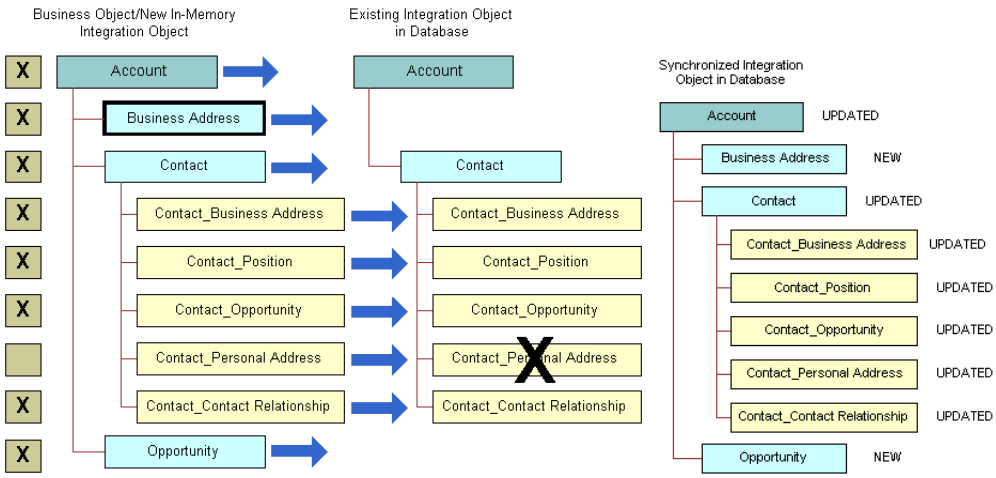


The integration object now contains two new components, *Business Address* and *Opportunity*. Other components are updated with the definitions of the corresponding components in the business object.

Deleting a Component from the Integration Object

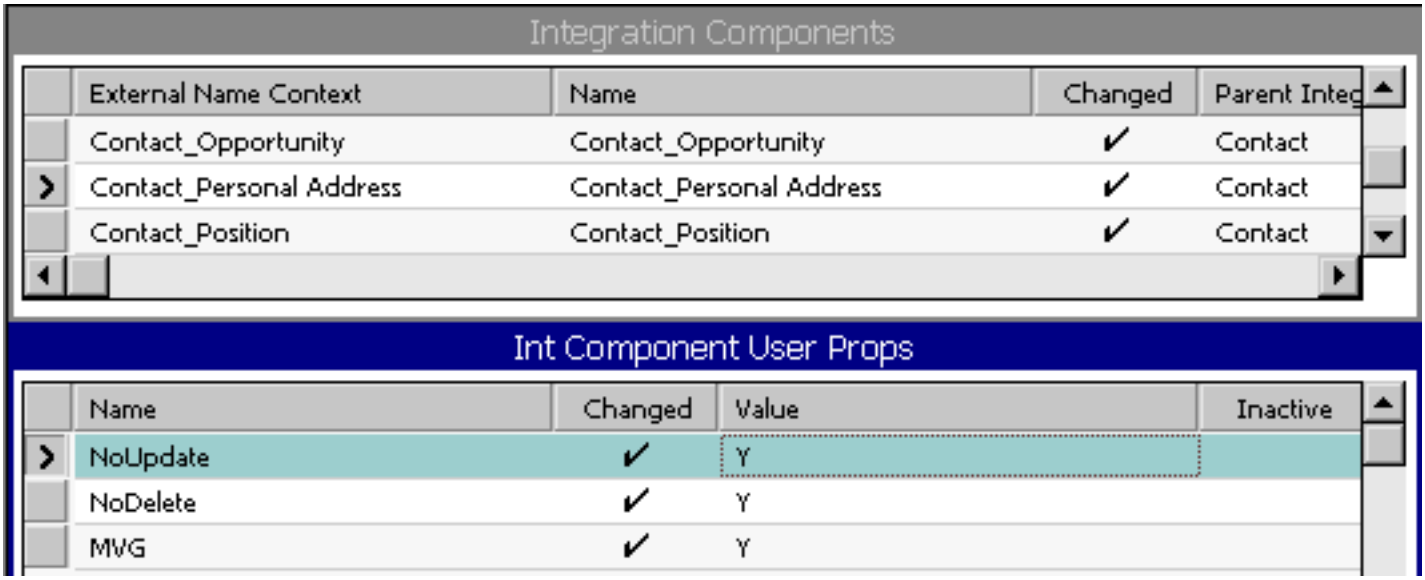
If you choose to deselect a component in the Synchronization wizard, then you specify to the wizard to delete the component in the integration object with the matching External Name Context property. The integration object that exists in the database has a component with the same External Name, External Name Sequence, and External Name Context as the unchecked component in the component selection tree.

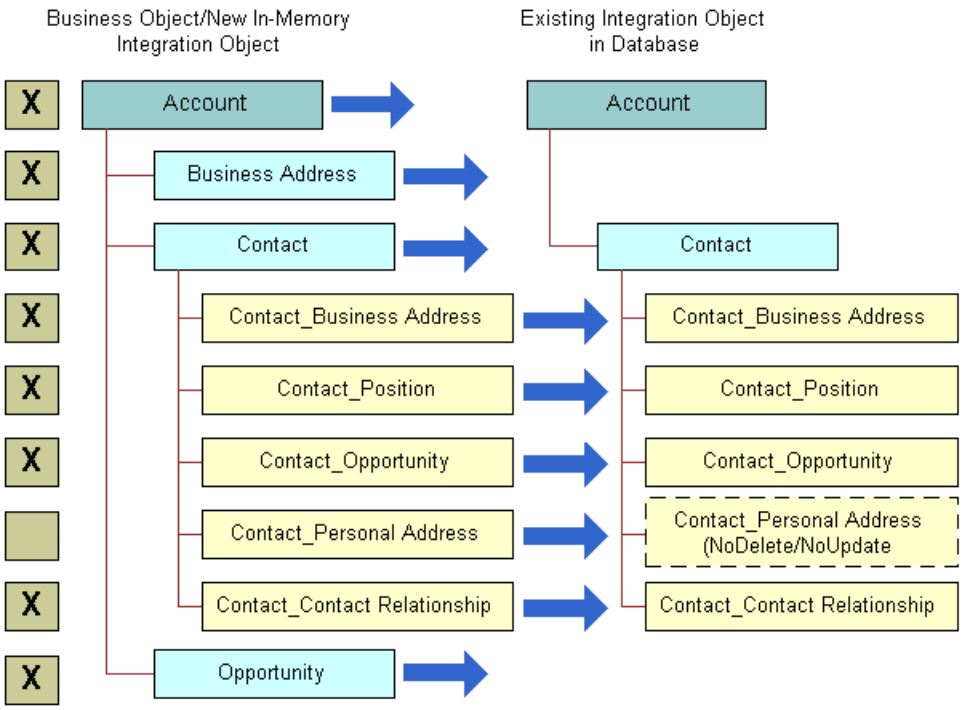
In the following figure, the Contact_Personal Address in the existing Account integration object is unchecked in the Synchronization wizard tree. This is represented by an *X* in this figure. The following figure illustrates this concept. The second figure in this topic shows the integration object after synchronization.

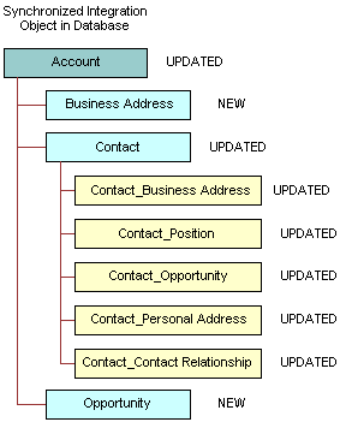


The component Contact_Personal Address has been deleted. When you use the updated integration object, you cannot pass data for that component between a Siebel application and an external application. This example shows you how you might cause unexpected results by deselecting components. However, if you do want to delete a particular component from the integration object, then deleting a component from the integration object method accomplishes that goal.

Guidelines for Maintaining Integration Objects







As the examples illustrate, you must be aware of the possible changes that can occur when you synchronize business objects and integration objects. The Integration Objects Synchronize wizard can provide assistance in managing your integration objects, but you must have a clear understanding of your requirements, your data model, and the Siebel business object structure before undertaking a task as important as synchronization.

To make maintenance of integration objects easier, adhere to the following guidelines when creating or editing your integration objects:

- Use a meaningful name for any user key you have added that is different from the generated user keys. Using meaningful names helps with debugging.
- Inactivate user keys instead of deleting them.
- Inactivate fields instead of deleting them.

Synchronizing Integration Objects

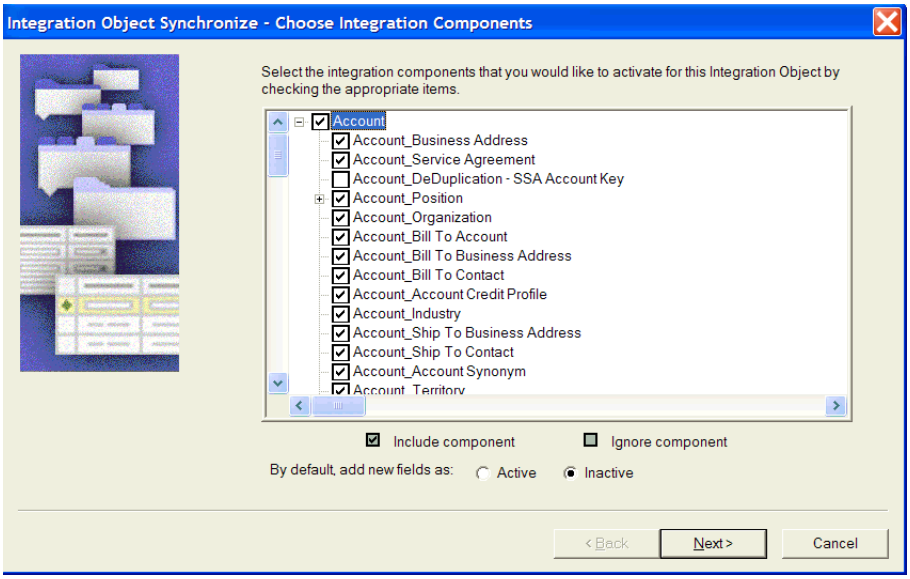
You use the Integration Object Synchronize wizard in Siebel Tools to update and synchronize integration objects.

Note: The update process overrides the integration object and deletes user keys, user properties, and so on. You can use the copy of the integration object made by the Synchronization wizard to see how you have modified the object.

To update an integration object with updated business object definitions

1. In Siebel Tools, select the integration object you want to update.
2. Click Synchronize in the Integration Objects list.

   The Integration Object Synchronize wizard appears.
3. Click on the plus sign to list all the related integration components, as shown in the following figure.



The process of retrieving Siebel integration objects and Siebel business object definitions can take varying amounts of time, depending on the size of the selected objects.

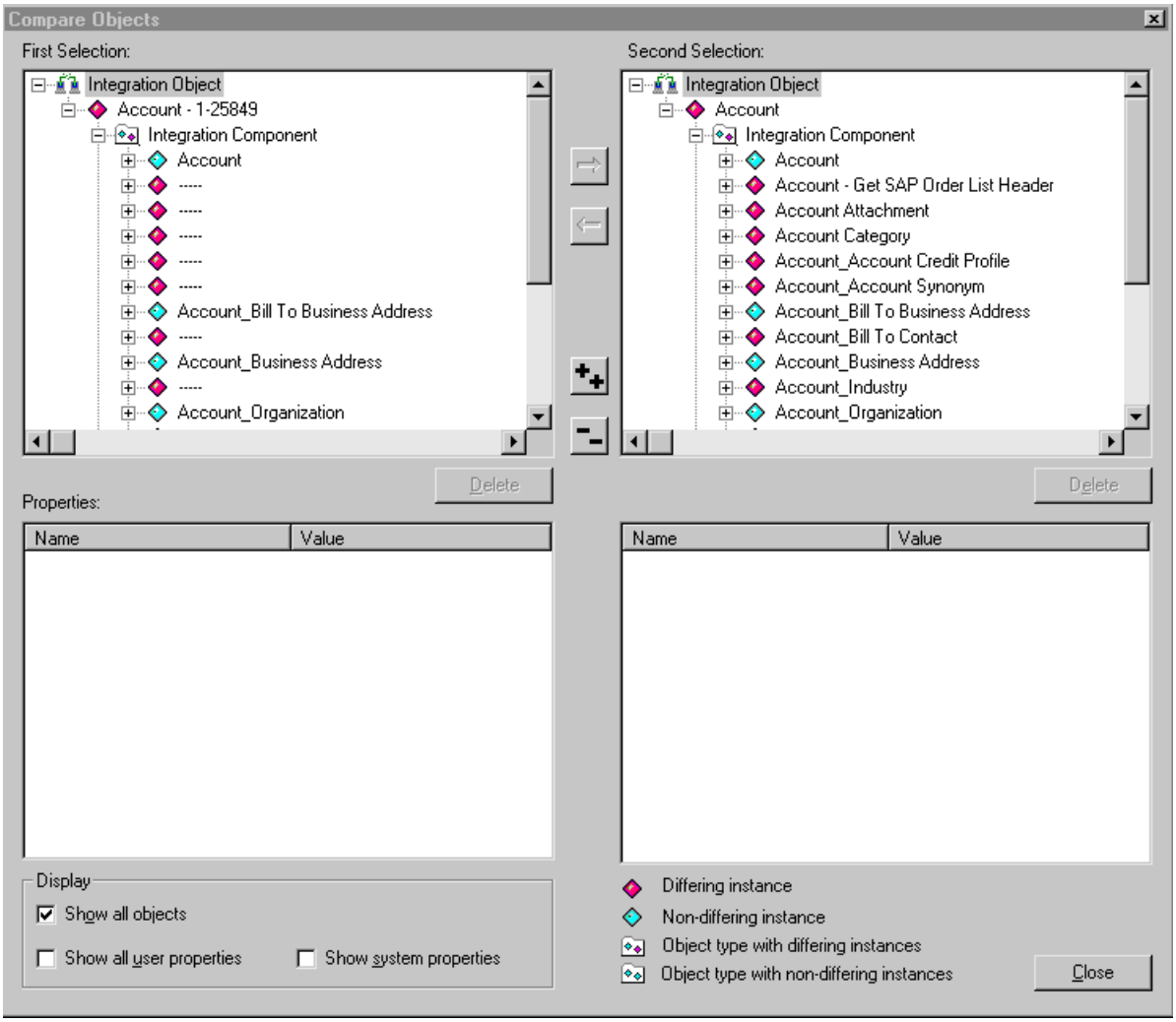
4. Uncheck the boxes beside the objects and components you do not want to include in the synchronization of your Siebel integration object. Note that only the objects that are included in the new integration object are marked.
5. Choose to add new fields as active or inactive and click Next. Inactive is the default.

   The process of performing the synchronization can take some time, depending on the complexity of the selected objects.

   The Integration Object Synchronize Summary screen appears, providing feedback from the synchronization.

   Each added field is checked as to whether or not it is required for use with the integration object.
6. Review the summary. If changes are needed, then click Back and make the needed changes.
7. If no changes are needed, then click Finish to synchronize the Siebel integration object and the Siebel business object.

   The Compare Objects dialog box appears. This tool allows you to move properties and objects between versions using arrow buttons.



When you synchronize the Siebel integration object and the Siebel business object, the Synchronization wizard performs update, insert, and delete operations on the existing integration object definition. The Synchronization wizard selects or deselects components to make the Siebel integration object look like the definition of the Siebel business object you chose.

The wizard generally updates the Siebel integration object either by updating the object and its components or by updating some components and deleting others. For information, see *Updating the Entire Integration Object* and *Deleting a Component from the Integration Object*.

8. Copy custom properties and custom user keys as needed. The wizard includes any new fields added to the business object in your integration object for the new version of your Siebel application. All these fields are set to active.
9. Inactivate any new fields that you do not need in a component of your updated integration object.
10. Right-click on your integration object, and select the Validate option to validate your integration object.

Note: If you want to synchronize any of the external integration objects, then follow this general procedure to perform a synchronization operation.

Resolving Synchronization Conflicts for Integration Objects and User Properties

This topic serves as a guide to resolving synchronization conflicts if any arise.

The following table illustrates the behavior of the merging logic for each of the integration object parts that have to be synchronized.

Integration Object Metadata	Merging Rules
Objects	Validate that Business Object still exists.
Components	Present the tree of components based on current business object definition. The components present in the current integration object are checked in the UI tree, other components presented as Inactive. User decides which components to add or delete. This is done by the Synchronization wizard UI.
Fields	Keep the current integration component fields if still present in the business component, otherwise delete. Add new fields in a way that does not conflict with existing ones (see the row about External Sequence for more information). System fields are created when appropriate (for example, searchspec, IsPrimaryMVG, and operation). If the system field is inconsistent with the integration component definition, then delete it. Active/Inactive. Preserve the current integration component field value unless Business Component Field is Required (field must be present during Insert). Otherwise, new fields are created Inactive.
XML Properties	Preserve the current integration object values to keep XML compatible. Add new components/fields properties avoiding conflict with existing XML. XML Properties are processed according to the XML sequence. New components/fields that sequence within the parent component element will be higher than current. Reuse existing processing code (and check for correct behavior).
External Sequence (on components or fields)	Preserve the component or field sequence within the parent component. Set the sequence on new components or fields higher than the existing ones.
Name	Preserve Names in the current integration object.
User key, Hierarchy key, Other keys (for example, Status Key)	Existing Keys: • Keep existing keys as Active if all the key fields are Active. • Keep existing keys Inactive if Inactive already or make Inactive if any of the fields are Inactive. • If a field is Inactive in an integration component, then make it Inactive in the key. Make the key Inactive. • If a field is not present in an integration component, then delete it from the key. Make the key Inactive.

Integration Object Metadata	Merging Rules
	New Keys: • Create new keys as Inactive. • If any of the key fields are Inactive, then either: • Do not create the key. • Make fields Active in the integration component.
User Properties	Preserve valid cases, remove invalid ones, and generate warnings. See the following table for more information.

The following table shows the logic that is used when synchronizing user properties.

User Property Name	Values (Default is in *italics*)	Level (Object, Component, or Field)	Merging Rules
AdminMode	Y, *N*	C, O	Entered by the user; if the value exists, then keep it. Otherwise, the wizard sets the value to N.
AllLangIndependentVals	Y,*N*	O	Entered by the user; if the value exists, then keep it. Otherwise, the wizard sets the value to N.
AssocFieldName	Any valid field name in the Association business component	F	Siebel Wizard generates the value based on current business component definition. The Wizard overwrites the user change, because in order for the integration component to be functional, the User Property has to be consistent with the business component. (component MVGAssociation is set to Y)
Association	Y, *N*	C	Siebel Wizard generates the value based on current business component definition. The Wizard overwrites the user change, because in order for the integration component to be functional, the User Property has to be consistent with the business component.
EDEnabled	Y, *N*	F	For each integration component field, the Synchronize wizard adds an integration component field user property named EDEnabled with the value set to Y if the corresponding business component field is effective dating enabled. The wizard will not overwrite user changes.
FieldDependency*FieldName*	Any active integration component name within the same integration component	F	Entered by the user. Keep the current value if valid (if *FieldName* field is Active).
ForceUpdate	Y, *N*	O	Entered by the user. Keep the current value.

User Property Name	Values (Default is in *italics*)	Level (Object, Component, or Field)	Merging Rules
Ignore Bounded Picklist	Y, *N*	O, C, F	Entered by user, keep if valid (if component Picklist is set to Y).
IgnorePermissionErrorsOnUpdate, IgnorePermissionErrorsOnInsert, IgnorePermissionErrorsOnDelete	Y, *N*	C	Entered by the user. Keep the current value.
MVG	Y, *N*	C	Siebel Wizard generates the value based on the current business component definition. The Wizard overwrites the user change, because in order for integration component to be functional, the User Property has to be consistent with the business component. IsPrimaryMVG system field is created in the merged integration object.
MVGAssociation	Y, *N*	C	Siebel Wizard generates the value based on the current business component definition. The Wizard overwrites the user change, because in order for integration component to be functional, the User Property has to be consistent with the business component. IsPrimaryMVG system field is created in merged integration object.
MVGFieldName	Any valid field name in the MVG business component	F	Siebel Wizard generates the value based on current business component definition. The Wizard overwrites the user change, because in order for integration component to be functional, the User Property has to be consistent with the business component. (component MVG is set to Y)
NoInsert, NoDelete, NoUpdate, NoQuery, NoSynchronize	Y, *N*	C, F (NoUpdate)	Entered by the user. Keep the current value.
Picklist	Y, *N*	F	Siebel Wizard generated. The user change is kept if valid (if Picklist component). Review the input object for a user property of PICKLIST. Copy from the current field.
PicklistUserKeys	Any active fields	F	Entered by user, keep only Active fields. User property is valid only if PICKLIST is set to Y on the integration component. If no Active fields left, then remove the user property.
SuppressQueryOnInsert	Y, *N*	C	Entered by the user. Keep the current value. When using the Insert method for the EAI Siebel Adapter, if this integration component user property is

User Property Name	Values (Default is in *italics*)	Level (Object, Component, or Field)	Merging Rules
			defined, then the EAI Siebel Adapter will not perform a query before inserting a record.
ViewMode	*All*, Manager, Sales Rep, and any others	O	Entered by the user; if the value exists, then keep it. Otherwise, the wizard sets the value to All.

Using Formatted Values in Integration Objects

The UseFormattedValues integration object user property allows you to configure the EAI Siebel Adapter to use formatted values.

The Siebel application stores a phone number's format as well as the phone number itself in the Siebel Database to display the phone number in the GUI. How a phone number displays is dependent on the preconfigured format for a specific country.

For example, a +55 555 5555 phone number in an English - ENU database is stored as +555555555 000 0000, where 000 0000 is the formatting mask.

By default, the value of the UseFormattedValues user property is set to N, indicating no formatted values are used. However, you can use scripting or a workflow to configure the EAI Siebel Adapter to force the use of formatted values by setting an integration object's UseFormattedValues user property to Y. In the previous example, the EAI Siebel Adapter will then return the phone number as +55 555 5555, ignoring the zeroes.

Note: UseFormattedValues is set at the integration object level and applies to all formattable fields in the integration object.

Generating Integration Object Schemas

At certain points in your integration project, you might want to generate schemas from an integration object. If you export Siebel integration objects as XML to other applications, then you might have to publish the schemas of such objects so that other applications can learn about the structure of the XML to expect.

To generate an integration object schema

1. In Siebel Tools, select the integration object for which you want to generate a schema.
2. Click Generate Schema to access the Generate XML Schema wizard.
3. Choose the business service to use to generate the schema:
    - **EAI XML DTD Generator.** Generates a Document Type Definition (DTD).
    - **EAI XML XDR Generator.** Generates an XML-Data Reduced (XDR) schema.

- **EAI XML XSD Generator.** Generates an XML Schema Definition (XSD).

4. Choose an envelope type to use in the schema, either none or Siebel Message Envelope.
5. Choose a location where you want to save the resulting schema file.
6. (Optional) Select the Include length information for String type check box to generate simple types for all string elements in the integration object schema.
7. Click Finish.

The wizard generates the selected type of schema for the integration object. Use this to help you map external data directly to the integration object. The schema serves as the definition for the XML elements you can create using an external application or XML editing tool.

Note: With the EAI XML DTD Generator, elements that appear more than once in the integration object structure are forward declared in the schema. A list of shared elements is generated, for example:

```
<!-- Shared Element List. These elements are guarenteed -->
<!-- to have the same datatype, length, precision, and scale.-->
<!ELEMENT ErrorMessage (#PCDATA) >
<!ELEMENT ErrorCode (#PCDATA) >
```

Optimizing the Performance of Integration Objects

To optimize your integration object performance, you might want to consider the following:

Size of Integration Object

The size of an integration object and its underlying business components can have an impact on the latency of the EAI Siebel Adapter operations. Inactivate unnecessary fields and components in your integration objects.

Force-Active Fields

Reexamine any fields in the underlying business component that have the force-active specification. Such fields are processed during the integration even if they are not included in the integration component. You might want to consider removing the force-active specification from such fields, unless you absolutely need them.

Picklist Validation

Siebel CRM has two classes of picklists: static picklists based on lists of values and dynamic picklists based on joins.

Setting the property PICKLIST to Y in the integration object field directs the EAI Siebel Adapter to validate that all operations conform to the picklist specified in the field. For dynamic picklists, this setting is essential to make sure the joins are resolved properly. However, for unbounded static picklists, this validation might be unnecessary and can be turned off by setting the PICKLIST property to N. Even for bounded static picklists, you can turn off validation in the adapter, because the Object Manager can perform the validation. Turning off the validation at the EAI Siebel Adapter level means that picklist-related warnings and debugging messages do not show up along with other EAI Siebel Adapter messages. This also means that bounded picklist errors will not be ignored, even if Ignore Bounded Picklist is set to Y.

As well as certain warnings and messages not appearing, setting the integration component field user property PICKLIST to N can also cause fields to be auto-completed. Providing only part of the value for a particular field causes the field to be auto-filled with the first matching entry in the picklist. This occurs especially when the picklist is based on a multilingual list of values (MLOV). For example, if the incoming message contains the string "On-" and there exists an entry "On-Hold," then the field will be set to "On-Hold."

If the EAI Siebel Adapter performs the validation (PICKLIST is set to Y), auto-filling of the field does not occur. In this case, the EAI Siebel Adapter supports only an exact match for the particular field (in the previous example, the value "On-" will fail; only "On-Hold" will pass).

Note: Performing the validation of a bounded picklist in the EAI Siebel Adapter is about 10% faster than performing the validation in the Object Manager.

About Business Component Restrictions for Integration Components

The business components underlying the Integration Components might have certain restrictions. For example, only an administrator can modify the Internal Product. The same restrictions apply during integration. In many cases, the Siebel Integration Object Builder wizard detects the restrictions, and sets properties such as No Insert or No Update on the integration components.

System Fields

Integration object fields marked as System are not exported during a query operation. This setting prevents the EAI Siebel Adapter from treating the field as a data field, which means for the Query and QueryPage method the EAI Siebel Adapter do not write to the field. For the Synchronize and Update method, the field will not be directly set in the business component unless the ISPrimaryMVG is set to Y.

If you want to include System fields in the exported message, then change the Integration Component field type to Data.

Note: System fields are read-only. If you attempt to send a message with the value set for a System field, then the setting will be ignored and a warning message will be logged. However, in order to permit updates for the System field SSA Primary Field, change the field type from System to Data. (SSA Primary Field is a pseudo-field that is used to mark the current instance of the child Integration Component to be a primary on the link from the parent component.)

Guidelines for Using Integration Components

The following are the guidelines for using integration components:

- Familiarize yourself with the business logic in the business components. Integration designers must use the presentation layer, or the user interface, to get a good sense of how the business component behaves, and what operations are allowed and not allowed.

- Design with performance in mind. For more information on performance and using integration objects, see *Optimizing the Performance of Integration Objects*.
- Design with maintenance in mind. For more information on maintenance, see *Guidelines for Maintaining Integration Objects*.
- Resolve configuration conflicts. During the development of your integration points, you might encounter issues with the configuration of business components that are configured to support interactive GUI usage, but do not satisfy your integration requirements.

  The following scenarios demonstrate two situations in which you might encounter such conflicts, and a possible solution for each case:

  Scenario 1. A business component such as Internal Product is made read-only for regular GUI usage, but you want your integration process to be able to update the Internal Product business component.

  Solution. Set the AdminMode user property on the integration object to Y. This allows the EAI Siebel Adapter to use the business component in an administrator mode.

  Scenario 2. Similar to scenario 2, a business component such as Internal Product is made read-only for regular GUI usage, but you want your integration process to be able to update the Internal Product business component. The only difference in this scenario is that the business component is used through a link that has NoUpdate property set to Y.

  Solution. Because there is a link with NoUpdate property set to Y, setting the AdminMode user property on the integration object to Y is not going to help. You must create the following exclusively for integration purposes:

  - A new link based on the original link with NoUpdate property Set to N.
  - A copy of the original business object referencing the new link instead of the original. Note that both links must use the same business component.

  > **Note:** Customized configurations are not automatically upgraded during the Siebel Repository upgrade, so use this option as a last resort.

4 Business Services

Business Services

This chapter outlines the basic concepts of a business service, its structure and purpose, and how you can customize and create your own business service. This chapter also describes how to test your business service before it is implemented. The following topics are included:

- *About Business Services*
- *Creating Business Services in Siebel Tools*
- *Creating Business Services in the Siebel Application*
- *Deploying Business Services as Web Services*
- *Exporting and Importing Business Services in Siebel Tools*
- *Importing Business Services into Siebel CRM*
- *Testing Your Business Service in the Simulator*
- *About Accessing a Business Service Using Siebel eScript or Siebel VB*
- *Business Scenario for the Use of Business Services*

About 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 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, a shipping rate calculation, or other specialized functions.

The business service can be assessed either directly by way of workflows (business processes) or by way of a scripting service written in Siebel VB or Siebel eScript.

The following topics are also covered:

- *About Creating Business Services*
- *Business Service Structure*
- *Property Sets*

About Creating Business Services

A Siebel application provides a number of prebuilt business services to assist you with your integration tasks. These services are based on specialized classes and are called Specialized Business Services. Many of these are used internally to manage a variety of tasks.

CAUTION: As with other specialized code such as Business Components, use only the specialized services that are documented in the *Siebel Bookshelf* . The use of undocumented services is not supported and can lead to undesired and unpredictable results.

Note: The *Siebel Bookshelf* is available on Oracle Technology Network (OTN), Oracle Software Delivery Cloud, or it might be installed locally on your intranet, or on a network location.

In addition to the prebuilt business services, you can build your own business service and its functionality in two different ways to suit your business requirements:

- **In Siebel Tools.** Created at design time in Siebel Tools using Siebel VB or Siebel eScript. Design-time business services are stored in the Siebel design time repository, so you have to compile the repository before testing them. When your test is completed, deliver the changes. The business services stored in the repository automatically come over to the new repository during the upgrade process. General business services are based on the class CSSService. However, for the purposes of Siebel EAI, you base your data transformation business services on the CSSEAIDTEScriptService class. For information, see *Creating Business Services in Siebel Tools*.
- **In Siebel client.** Created at run time in the Siebel client using the Business Service Administration screens. Run-time business services are stored in the Siebel run-time database, so they can be tested right away. The run-time business services have to be migrated manually after an upgrade process. For information, see *Creating Business Services in the Siebel Application*.

Note: To use the DTE scripts, write your business service in Siebel eScript; otherwise, you can write them in Siebel VB.

Business Service Structure

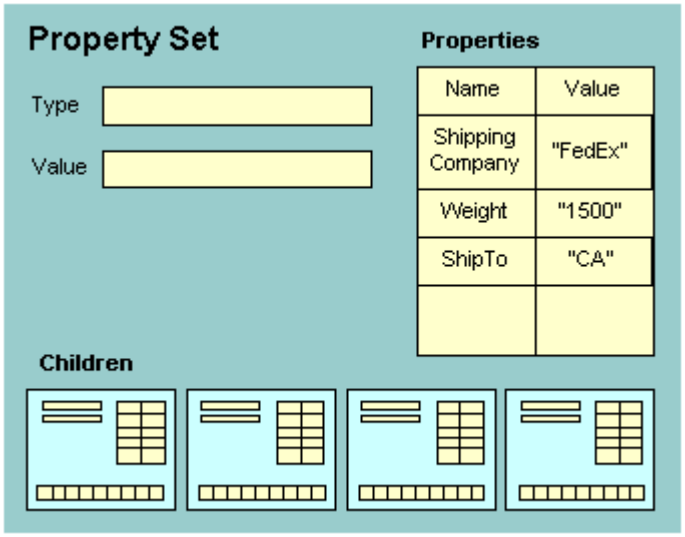
Business services allow developers to encapsulate business logic in a central location, abstracting the logic from the data it might act upon. A business service is much like an object in an object-oriented programming language.

A business service has properties and methods, and maintains a state. Methods take arguments that can be passed into the object programmatically or, in the case of Siebel EAI, declaratively by way of workflows.

Note: For more information on business service methods and method arguments, see *Siebel Object Interfaces Reference* .

Property Sets

Property sets are used internally to represent Siebel EAI data. A property set is a logical memory structure that is used to pass the data between business services. The following figure illustrates the concept of a property set.



The property set consists of four parts:

- **Type.** Used to describe what type of object is being represented.
- **Value.** Used to hold serialized data, such as a string of XML data.

  > **Note:** In Siebel Tools, a Value argument to a method is shown with the name of <Value> , including the angle brackets. You can also define a Display Name for the Value argument in Siebel Tools. This Display Name appears in the Siebel Business Process Designer when you are building integration workflows. In this guide, the Display Name Message Text is shown when referring to the Value argument, and the Name <Value> is shown when referring to the *Value* of the Value argument.

- **Properties.** A table containing name-value pairs. You can use the properties to represent column names and data, field names and data, or other types of name-value pairs.
- **Children.** An array of child-level property sets. You can use the array to represent instances of integration objects. For example, a result set might contain an Account with some set of contact records from the database. Each contact record is represented as a child property set.

It is recommended that you treat input property sets in business services as constants. If you must modify the inputs, then make a copy first. Otherwise, there might be interference between business service scripts and workflows that also modify the inputs, leading to unpredictable application behavior.

For example, when creating the XMLHierarchy property set using a custom business service in a workflow, if the input property set is modified without making a copy, then the following error occurs:

```
Argument 'XMLHierarchy' in step 'Convert XMLHierarchy' is not correctly initialized
or does not return valid data.(SBL-BPR-00107)
```

> **Note:** For more information on property sets and their methods, see *Siebel Object Interfaces Reference* .

Creating Business Services in Siebel Tools

The following procedures explain how to create business services and business service scripts in Siebel Tools:

- *Defining a Business Service in Siebel Tools*.
- *Defining Business Service Methods*
- *Defining Business Service Method Arguments*
- *Writing Business Service Scripts*
- *Defining Business Service User Properties*

Note: Business services you create in Siebel Tools must be delivered.

Defining a Business Service in Siebel Tools

You declaratively define the business service in Siebel Tools, and then add your scripts to the business service in the Siebel Script Editor within Siebel Tools.

To define a business service in Siebel Tools

1. In Siebel Tools, select and lock the project with which you want to associate your business service.

   Note: Each business service must belong to a project, and the project must be locked. For more information, see *Using Siebel Tools* .

2. Select the Business Services object in the Tools Object Explorer.

   The list of predefined business services appears in the farthest panel.
3. Right-click, and then choose New Record.
4. Type a name in the Name field of the new business service.
5. From the pull-down menu in the Project field, pick the project you locked in Step 1.
6. Choose the appropriate class for your business service from the Class picklist:
   - Data transformation business services must use the CSSEAIDTEScriptService class.
   - Other business services will typically use the CSSService class.
7. Step off the current record to save your changes.

Defining Business Service Methods

Business services contain related methods that provide the ability to perform a particular task or set of tasks.

Note: For information on business service methods, see *Siebel Object Interfaces Reference* .

To define a business service method

1. With your business service selected in Siebel Tools, expand the Business Service tree in the Object Explorer, and then select Business Service Method.

   The Business Service Methods list appears in the Object List Editor. If you have already defined methods for the selected business service, they appear in the Business Services Methods list.
2. Right-click, and then choose New Record.
3. Type the name of the method in the Name field of the new method.

Defining Business Service Method Arguments

Each method can take one or more arguments. The argument is passed to the method and consists of some data or object that the method processes to complete its task.

To define business service method arguments

1. With your business service method selected in Siebel Tools, expand the Business Service Method tree in the Object Explorer, and then select Business Service Method Args.

   The Business Service Methods Args list appears in the Object List Editor.
2. Right-click, and then choose New Record.
3. Type the name of the argument in the Name field of the new method argument record.

   > **Note:** If you plan to use this business service in a Siebel CRM application, then specify the Display Name as well.

4. Enter the data type in the Data Type field.
5. Check the Optional check box if you do not want the argument to be required for the method.
6. Choose a Type for the argument. Refer to the following table for a list of different types and their descriptions.

Argument	Description
Input	This type of argument serves as input to the method.
Input/Output	This type of argument serves as both input to the method and output from the method.
Output	This type of argument serves as output from the method.

Writing Business Service Scripts

Business service scripts supply the actual functionality of the business service in either Siebel VB or Siebel eScript. As with any object, the script you provide is attached to the business service.

To write business service scripts

1. In Siebel Tools, select the business service for which you want to write a script.
2. Right-click, and then choose Edit Server Scripts.

   The Siebel Script Editor appears.
3. Select either Siebel eScript or Visual Basic for your scripting language.
4. Select Service_PreInvokedMethod as the event handler.

   > **Note:** To write any Siebel VB script in the Business Services, the operating system you are using must support Siebel VB. Siebel VB is not supported in the UNIX environments.

5. Type your script into the Script Editor.

   > **Note:** Write your business service in Siebel eScript if you want to use the DTE scripts. For information on scripting, see *Using Siebel Tools* .

Defining Business Service User Properties

User properties are optional variables that you can use to define default values for your business services in Siebel Tools. When a script or control calls your business service, one of the first tasks the service performs is to check the user properties to gather any default values that will become input arguments to the service's methods.

To define business service user properties

1. With your business service selected in Siebel Tools, expand the Business Service tree in the Object Explorer, and then select Business Service User Prop.

   The Business Service User Props list appears in the Object List Editor.
2. Right-click, then choose New Record.
3. Type the name of the user property in the Name field of the new record.
4. Type a value in the Value field.

   The value can be an integer, a string, or a Boolean.

Creating Business Services in the Siebel Application

You can define business services in the Siebel application using the Business Service Administration screens. The business services you create in the client are stored in the Siebel Database. This topic illustrates the creation of business services using the Business Service Methods view, which includes applets to create and display the business service.

To define a business service in the Siebel application

1. Navigate to the Administration - Business Service screen, Methods view.
2. Click New to create a new record in the Methods form applet:
   - **Name.** Name of the business service.

- **Cache.** If checked then the business service instance remains in existence until the user's session is finished; otherwise, the business service instance will be deleted after it finishes executing.
- **Inactive.** Check if you do not want to use the business service.

3. Define methods for the business service in the Methods list applet:
   - **Name.** Name of the method.
   - **Inactive.** Check if you do not want to use the method.
4. Define method arguments for the methods in the Method Arguments list applet:
   - **Name.** Name of the method argument.
   - **Type.** The type of the business service method argument. Valid values are Output, Input, and Input/Output.
   - **Optional.** Check if you do not want this argument to be optional.
   - **Inactive.** Check if you do not want to use the argument.
5. From the link bar, select Scripts.
6. Write your Siebel eScript or VB code in the Business Service Scripts list applet.

   Note: To write any Siebel VB script in the Business Services, the operating system you are using must support Siebel VB. Siebel VB is not supported in UNIX environments.

7. Click Check Syntax to check the syntax of the business service script.

Deploying Business Services as Web Services

You can deploy business services, which you have created in Siebel Tools, as Web services. The Web services can then be consumed by other applications.

To be deployed, a business service must have at least one accessible method that is supported in Siebel inbound Web services. The business service must include a valid integration object name for any hierarchical argument.

Note: The Hierarchy type is not supported unless a valid integration object name is specified.

For more information on Web services, see *Web Services*.

To deploy a business service as a Web service

1. In the Object Explorer in Siebel Tools, select the Business Service object.
   The Business Services list appears.
2. In the Object List Editor, right-click the business service to deploy, and then choose Deploy as Web Service.
   The Expose Business Service as Web Service dialog box appears.
3. Specify the following in the dialog box, and then click Finish:
   - **Business service methods to expose.** The operation names for the business service methods are system generated. To edit an operation name, click it in the list.
   - **URL for the Web service.** Replace `<webserver>` with a valid host name and `<lang>` with a valid language code, such as `enu`.

- **Generate WSDL check box.** To generate a Web Services Description Language (WSDL) file, select the check box, and then choose a location to save the WSDL file.

The business service is deployed. Deployed business services are shown in the Administration - Business Services screen in the Siebel client. Deployed Web services are shown in the Administration - Web Services screen, Inbound Web Services view.

You can also remove (undeploy) deployed business services from the Siebel run-time database.

To undeploy a business service

1. In the Siebel client, navigate to the Administration - Business Services screen.

   The Details list appears.
2. Query for the deployed business service, and then select it.
3. Click Delete.

The business service is undeployed.

Exporting and Importing Business Services in Siebel Tools

You can export business services into an XML file by clicking Export in the Business Services list in the Object List Editor. This writes the definition of the business service, including every method, method argument, and script, into the XML file.

You can import a business service from an external XML file by clicking Import in the Business Services list in the Object List Editor.

Importing Business Services into Siebel CRM

You can import business services, which you have created in Siebel Tools and exported as XML files, into the Siebel run-time database. This saves time by allowing you to modify business service definitions without having to shut down your production environment, edit the business services in workspace of Siebel Tools or Web Tools, and then deliver the workspace.

To import a business service into Siebel CRM

1. Navigate to the Administration - Business Service screen, Details view.
2. From the Menu pull-down, choose Import Service.
3. The Business Service Import dialog appears.
4. Browse for a business service XML file, and then click Import.

Testing Your Business Service in the Simulator

You can use the Business Service Simulator to test your business services in an interactive mode.

To run the Business Service Simulator

1. Navigate to the Administration - Business Service screen, Simulator view.

   > **Note:** The contents of the Simulator view are not persistent. To save the data entered in the applets, click the Save To File button. This will save the data for the active applet in an XML file. The data can then be loaded into the next session from an XML file by clicking on the Load From File button.

2. In the Simulator list applet, click New to add the business service you want to test.
3. Specify the Service Name and the Method Name.
4. Enter the number of iterations you want to run the business service:
   - Specify the input parameters for the Business Service Method in the Input Property Set applet. Multiple input property sets can be defined and are identified by specifying a Test Case #.
   - If the Input Property Set has multiple properties, then these can be specified by clicking on the glyph in the Property Name field. Hierarchical property sets can also be defined by clicking on the glyph in the Child Type field.
5. Click Run to run the business service.

   The Simulator runs the specified number of iterations and loops through the test cases in order. If you have defined multiple input arguments, then you can choose to run only one argument at a time by clicking Run On One Input.

   The result appears in the Output Property Set applet.

   > **Note:** When the Output arguments are created, you can click Move To Input to test the outputs as inputs to another method.

About Accessing a Business Service Using Siebel eScript or Siebel VB

In addition to accessing a business service through a workflow, you can use Siebel VB or eScript to call a business service. The following Siebel eScript code calls the business service EAI XML Read from File to read an XML file, and produces a property set as an output. The EAI Siebel Adapter uses the output property set to insert a new account into the Siebel application:

```
var svcReadFile = TheApplication().GetService("EAI XML Read from File") ;
var svcSaveData = TheApplication().GetService("EAI Siebel Adapter");
var child = TheApplication().NewPropertySet();
var psInputs = TheApplication().NewPropertySet();
var psOutputs = TheApplication().NewPropertySet();
var psOutputs2 = TheApplication().NewPropertySet();
var svcSaveData = TheApplication().GetService("EAI Siebel Adapter");
```

```
psInputs.SetProperty("FileName", "c:\\NewAccount.xml");
psOutputs.SetType "SiebelMessage";
psOutputs.SetProperty "IntObjectName","Sample Account";
psOutputs.SetProperty "MessageId", "";
psOutputs.SetProperty "MessageType", "Integration Object";
svcReadFile.InvokeMethod("ReadEAIMsg",psInputs, psOutputs);
svcSaveData.InvokeMethod("Upsert",psOutputs,psOutputs2);
```

The following Siebel VB sample code shows how to call the EAI File Transport business service to read an XML file. It also shows how to use the XML Converter business service to produce a property set:

```
Set Inp = TheApplication.NewPropertySet
Inp.SetProperty "FileName", "c:\test.xml"
Inp.SetProperty "DispatchService", "XML Converter"
Inp.SetProperty "DispatchMethod" , "XMLToPropSet"
Set svc = theApplication.GetService("EAI File Transport")
Set XMLOutputs = theApplication.NewPropertySet
svc.InvokeMethod "ReceiveDispatch", Inp, XMLOutputs
TheApplication.RaiseErrorText Cstr(XMLOutputs.GetChildCount)
```

Business Scenario for the Use of Business Services

Consider an example of a form on a corporate Web site. Many visitors during the day enter their personal data into the fields on the Web form. The field names represent arguments, whereas the personal data represent data. When the visitor clicks Submit on the form, the form's CGI script formats and sends the data by way of the HTTP transport protocol to the corporate Web server. The CGI script can be written in JavaScript, Perl, or another scripting language.

The CGI script might have extracted the field names and created XML elements from them to resemble the following XML tags:

```
First Name = <FirstName></FirstName>

Last Name = <LastName></LastName>
```

The CGI script might then have wrapped each data item inside the XML tags:

```
<FirstName>Hector</FirstName>

<LastName>Alacon</LastName>
```

To insert the preceding data into the Siebel Database as a Contact, your script calls a business service that formats the XML input into a property set structure that the Siebel application recognizes.

Code Sample Example for Creating a Property Set

The following is an example of the Siebel eScript code that you must write to create the property set:

```
x = TheApplication.InvokeMethod("WebForm", inputs, outputs);
var svc; // variable to contain the handle to the Service
var inputs; // variable to contain the XML input
var outputs; // variable to contain the output property set
svc = TheApplication().GetService("EAI XML Read from File");
 inputs = TheApplication().ReadEAIMsg("webform.xml");
 outputs = TheApplication().NewPropertySet();
svc.InvokeMethod("Read XML Hierarchy", inputs, outputs);
```

The following functions could be called from the preceding code. You attach the function to a business service in Siebel Tools:

Note: You cannot pass a business object as an argument to a business service method.

```
Function Service_PreInvokeMethod(MethodName, inputs, outputs)
 {
 if (MethodName=="GetWebContact")
 {
 fname = inputs.GetProperty("<First Name>");
 lname = inputs.GetProperty("<Last Name>");
 outputs.SetProperty("First Name",fname);
 outputs.SetProperty("Last Name", lname);
 return(CancelOperation);
 }
return(ContinueOperation);
}
Function Service_PreCanInvokeMethod(MethodName, CanInvoke)

{
 if (MethodName="GetWebContact")
 {
 CanInvoke ="TRUE";
 return (CancelOperation);
 }
 else
 {
 return (ContinueOperation);

  }
}
```

5 Web Services

Web Services

This chapter describes Web services, their uses, and how to create, implement, and publish Siebel Web services. This chapter also provides examples of how to invoke an external Web service and a Siebel Web service. The following topics are included:

- *About Web Services*
- *About RPC-Literal and DOC-Literal Bindings*
- *About One-Way Operations and Web Services*
- *Invoking Siebel Web Services Using an External System*
- *Consuming External Web Services Using Siebel Web Services*
- *Using the Local Business Service*
- *Using the Local Business Service in an Outbound Web Service*
- *Examples of Invoking Web Services*
- *About Web Services Security Support*
- *About Siebel Authentication and Session Management SOAP Headers*
- *About Web Services and Web Single Sign-On Authentication*
- *About SOAP Fault Schema Support*
- *About Custom SOAP Filters*
- *About EAI File Streaming*
- *About Web Services Cache Refresh*
- *Enabling Web Services Tracing*
- *Previewing the Repository Changes Before Delivery*

About Web Services

Web services combine component-based development and Internet standards and protocols that include HTTP, XML, Simple Object Access Protocol (SOAP), and Web Services Description Language (WSDL). You can reuse Web services regardless of how they are implemented. Web services can be developed on any platform and in any development environment as long as they can communicate with other Web services using these common protocols.

Business services or workflows in Siebel CRM can be exposed as Web services to be consumed by an application. Siebel Web Services framework has an ability to generate WSDL files to describe the Web services hosted by the Siebel application. Also, the Siebel Web Services framework can invoke external Web services. This is accomplished by importing a WSDL document, described as an external Web service, using the WSDL Import Wizard in Siebel Tools.

To specify the structure of XML used in the body of SOAP messages, Web services use an XML Schema Definition (XSD) standard. The XSD standard describes an XML document structure in terms of XML elements and attributes. It also specifies abstract data types, and defines and extends the value domains.

Users or programs interact with Web services by exchanging XML messages that conform to Simple Object Access Protocol (SOAP). For Web services support, SOAP provides a standard SOAP envelope, standard encoding rules that specify mapping of data based on an abstract data type into an XML instance and back, and conventions for how to make remote procedure calls (RPC) using SOAP messages.

Supported Web Services Standards

For information on the Web services standards supported with Siebel CRM, see the *Certifications tab on My Oracle Support*.

About RPC-Literal and DOC-Literal Bindings

In Siebel CRM, publishing a Siebel Web service as a Document-Literal (DOC-Literal) or RPC-Literal bound Web service partly conforms to the specification as defined by the Web Services Interoperability Organization's (WS-I) Basic Profile specification. Adherence to this specification makes sure that Siebel CRM can interoperate with external Web service providers.

WS-I is a trademark of the Web Services Interoperability Organization in the United States and other countries.

RPC-Literal Support

RPC allows the use of transports other than HTTP (for example, MQ and MSMQ), because you do not have to use the SOAPAction header to specify the operation.

For information on the Web services standards supported with Siebel CRM, see the *Certifications tab on My Oracle Support*.

Making a Web Service an RPC-Literal Web Service

RPC literal processing is enabled by rendering a Web service as an RPC-literal Web service, and choosing the correct binding on the Inbound Web Services view.

To make a Web service an RPC-literal Web service

1. Navigate to the Administration - Web Services screen, Inbound Web Services view.
2. Select or add a new namespace from the Inbound Web Services list following the instructions in *Invoking Siebel Web Services Using an External System*.
3. Create a new inbound service port record in the Service Ports list, as indicated in *Invoking Siebel Web Services Using an External System*
4. In the Binding column, select SOAP_RPC_LITERAL from the drop-down list.

DOC-Literal Support

When a SOAP DOC-literal binding is used, the SOAP envelope (the Body element) will contain the document WSDL part without any wrapper elements. The SOAP operation is determined by way of a SOAPAction HTTP header.

Note: SOAP:Body is in the instance SOAP message, but soapbind:body is the attribute in the WSDL document.

For information on the Web services standards supported with Siebel CRM, see the *Certifications tab on My Oracle Support*.

About One-Way Operations and Web Services

One-Way operations provide a means of sending a request to a Web service with the expectation that a SOAP response will not be returned. The Siebel application provides the ability to publish and consume Web services that implement one-way operations.

One-way operations come into play in both inbound and outbound scenarios:

- **Inbound.** If the Business Service Workflow method does not have any output arguments, then it is a one-way operation.
- **Outbound.** If the service proxy method has no output arguments, then it is a one-way operation.

Consider using one-way operations when data loss is tolerable. In cases involving one-way operations, you send a SOAP request and do not receive a SOAP response. The provider receives the SOAP request and processes it.

Note: It is important to note that SOAP faults, if any, are not returned as well.

Defining Support for One-Way Operations

For information on the Web services standards supported with Siebel CRM, see the *Certifications tab on My Oracle Support*.

Invoking Siebel Web Services Using an External System

The Siebel application allows enterprises to publish any business service or business process as a Web service. This process is also known as creating an inbound Web service. When the business service or business process is defined, a Siebel administrator navigates to the Administration - Web Services screen, Inbound Web Services view in the Siebel Web Client, and publishes it as a Web service. When the business service or business process is published as a Web service, the administrator generates the Web Service Definition Language (WSDL) document for the newly created Web service. The resulting WSDL document is consumed by an external application to invoke this Web service.

Note: You can deploy business services and workflows as Web services and generate WSDL files directly from Siebel Tools. For information on deploying business services, see *Deploying Business Services as Web Services*. For information on deploying workflows as Web services, see *Siebel Business Process Framework: Workflow Guide* .

The following inbound Web services topics are covered:

- *Publishing Inbound Web Services*
- *Generating a WSDL File*

- *About the Relationship of Port Types and Operations*
- *About Defining the Web Service Inbound Dispatcher*
- *Invoking Web Services on the Siebel Mobile Web Client*

Publishing Inbound Web Services

You can create and publish an inbound Web service using the Inbound Web Services view, as illustrated in the following procedure. You can then use the new inbound Web service when generating a WSDL document.

> **Note:** If publishing an ASI as an inbound Web service, then make sure that the ASI is enabled for external use in Siebel Tools.

To create an inbound Web service

1. Navigate to the Administration - Web Services screen, Inbound Web Services view.
2. In the Inbound Web Services list, create a new record:
   a. Enter the namespace for your organization's Web services in the Namespace column.

      > **Note:** This step is required for generating various XML documents.

   b. Enter the name of the inbound Web service in the Name column.
   c. Select Active in the Status field to enable external applications to invoke the Web service.

      > **Note:** If the Web service is inactive, then the external applications cannot invoke the Web service without clearing the cache.

   d. (Optional) Enter a description of the Web service in the Comment column.
3. Create an inbound service port record in the Service Ports list:
   a. Click New and enter the name of the port in the Name column.
   b. Pick the type of object published. If the required type is not available, then add a new type following Step c through Step f; otherwise, move to Step g.
   c. Click New and select the implementation type (Business Service or Workflow Process).
   d. Select the implementation name (the business service or workflow that implements the port type).
   e. Enter a name for the new type in the Name field and click Save.
   f. Click Pick in the Inbound Web Services Port Type Pick Applet to complete the process of adding a new Type.
   g. Select the protocol or transport that will publish the Web service.
   h. Enter the address appropriate for the transport chosen:
      - For the HTTP transport, enter an HTTP address of the Web service to be invoked, such as:

        ```
        http://mycompany.com/webservice/orderservice
        ```

      - For the JMS transport, enter the following:

        ```
        jms://YourQueueName@YourConnectionFactory
        ```

- For the Local Web Service transport, enter the name of the inbound port.
- For the EAI MSMQ Server transport, enter one of the following:

```
mq://YourQueueName@YourQueueManagerName
msmq://YourQueueName@YourQueueMachineName
```

> **Note:** With the EAI MQSeries, EAI MSMQ, and EAI JMS transports, the request and response must be in the same queue. When publishing using EAI MQSeries, EAI MSMQ, or EAI JMS, you cannot generate WSDL files.

i. Select the binding that will publish the Web service.

> **Note:** RPC_Encoded, RPC_Literal, and DOC_Literal styles of binding are supported for publishing Web services.

j. Enter a description of the Port in the Comment column.

4. In the Operations list, create a new operation record for the new service port:

> **Note:** Only the operations created in this step will be published and usable by applications calling the Web service. Other business service methods will not be available to external applications and can only be used for internal business service calls.

a. Enter the name of the Web service operation.
b. Select the name of the business service method in the Method Display Name column.

> **Note:** The Method Display Name column displays RunProcess by default if you chose Workflow Process as the type for your service port. However, you can change this to another name.

c. Select the authentication type from the drop-down list.

For more information on using the Username/Password Authentication Type, see *About RPC-Literal and DOC-Literal Bindings*.

Generating a WSDL File

The WSDL file specifies the interface to the inbound Web service. This file is used by Web service clients to support creation of code to call the Siebel Web service.

When you have created a new inbound Web service record you can generate a WSDL document, as described in the following procedure.

To generate a WSDL file

1. In the Inbound Web Services view, choose the inbound Web services you want to publish, and then click Generate WSDL.

   A WSDL file is generated that describes the Web service.
2. Save the generated file.
3. Import the WSDL to the external system using one of the following utilities:

- In Microsoft VisualStudio.Net, use the wsdl.exe utility, for example:

```
wsdl.exe /l:CS mywsdlfile.wsdl
```

- In Apache AXIS, use the wsdl2java utility, for example:

```
java org.apache.axis.wsdl.WSDL2Java mywsdlfile.wsdl
```

- In IBM WSADIE, depending on the version, add the WSDL file to the Services perspective and then run the Create Service Proxy wizard.
- In Oracle JDeveloper, use the Java Web Service from WSDL wizard.

> **Note:** These utilities only generate proxy classes. Developers are responsible for writing code that uses the proxy classes.

About the Relationship of Port Types and Operations

Port types are defined in the Inbound Web Services view, in the Service Ports applet. The Type and Business Service/ Business Process Name fields are based on the same dynamic picklist. Opening it displays all the port types. Here you can create or delete port types.

After a port type has been created, you can create the operations that the port type will define. This is done in the Operations applet. Clicking the New button displays any operations that are currently defined for the specified port type. You can expose as many business service methods as you want, but once defined they cannot be deleted or modified through the picklist or through the Operations applet. You can only delete the link between the specified port and operation.

The business service methods are read from the runtime repository. When an operation is defined, a new record is added to the S_WS_OPERATION table, with the Method Display Name field set to the business service method.

Subsequent attempts to add new operations display the dynamic picklist of operations stored in the S_WS_OPERATION table. Any changes to the business service definition made after the Web service operation was created are not reflected, because operations are read from the database.

When generating a WSDL, the generator reads the port type definition from the database and retrieves all associated operations. It processes the operations and then checks them against the business service methods in the runtime repository. Any discrepancy causes an error to be thrown.

This design allows port types to be shared across Web services. Changes to a port type (including the associated operations) made in one Web service definition do not affect other Web services. You can only make changes to a port type (such as deletion) after no Web services are pointing to it.

Deleting Operations by Deleting the Port Type

Operations themselves cannot be deleted after being created. The only way to delete an operation is to delete the associated port type.

> **Note:** Deleting a port type will cause all associated operations to be deleted.

To delete a port type and its operations

1. Delete all Service Port records that use this port type.
2. Click New to display the picklist.
3. Delete the port type, which will trigger the deletion of all associated operations.

About Defining the Web Service Inbound Dispatcher

The Web Service Inbound Dispatcher is a business service that is called by an inbound transport server component (or an outbound Web service dispatcher locally). This business service analyzes input SOAP messages containing XML data, converts the XML data to an XML hierarchy, maps the XML hierarchy to business service method arguments, and calls the appropriate method for the appropriate service (business service or process). After the called method finishes its execution, the Web Service Inbound Dispatcher converts the output arguments to XML data, and returns the XML embedded in the SOAP envelope. During this process, any errors are returned as SOAP fault messages.

SOAP Fault Message Example

When the code within a Web service raises an exception anywhere in the Web services stack, the exception is caught and transformed into a SOAP fault message.

For instance, the following example illustrates a particular case where mustUnderstand has been set to 1; and therefore, the header is interpreted as being mandatory. However, the corresponding filter and handler to process the header was not defined. This causes a SOAP fault message to be returned.

The format of the Siebel SOAP fault message for this example follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
- <SOAP-ENV:Body>
- <SOAP-ENV:Fault>
<faultcode>SOAP-ENV:MustUnderstand</faultcode>
<faultstring>Unable to process SOAP Header child element
'newns:AnotherUselessHeader' with 'mustUnderstand="1"'(SBL-EAI-08000)
</faultstring>
- <detail>
- <siebelf:errorstack xmlns:siebelf="http://www.siebel.com/ws/fault">
- <siebelf:error>
<siebelf:errorsymbol />
<siebelf:errormsg>Unable to process SOAP Header child element
'newns:AnotherUselessHeader' with 'mustUnderstand="1"'(SBL-EAI-08000)
</siebelf:errormsg>
</siebelf:error>
</siebelf:errorstack>
</detail>
</SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

For more information on SOAP fault handling, see *About SOAP Fault Schema Support*.

Invoking Web Services on the Siebel Mobile Web Client

The Siebel Mobile Web Client can serve the same Web services as those deployed on the Siebel Server, while protecting access through simple authentication. This feature allows developers to integrate external applications with Siebel CRM and test their integrations, without having to install an entire Siebel Enterprise.

> **Note:** All information provided in this topic for the Siebel Mobile Web Client also applies to the Siebel Developer Web Client.

The Web service functionality is an extension of the Siebel Mobile Web Client, and runs as a separate siebel.exe process. This second siebel.exe process is started by the Siebel Mobile Web Client as its child process. The child process listens on the specified port for all Web service requests. The Web service requests are processed and sent to the EAI Inbound Dispatch Service, and then the response is sent back to the Siebel Mobile Web Client. The child process exits when the Siebel Mobile Web Client exits.

> **Note:** If any changes are made to Web services in the run time, then these will not be available to the child process. You must restart the Siebel Mobile Web Client; you cannot clear the Web services cache.

Exceptions to Web Service Support

The Siebel Mobile Web Client provides the same Web service support as an EAI-enabled Siebel Server, with the following exceptions:

- The Web service consumer, such as soapUI, must be on the same computer as the Siebel Mobile Web Client.
- HTTPS is not supported.
- The Stateless, Stateful, and ServerDetermine session types are not supported. Only the None session type is supported.
- Concurrent requests are not serviced in parallel. There is only one siebel.exe process that serves Web services, so concurrent requests are queued.

  > **Note:** When multiple Siebel Mobile Web Client instances are running, there will not be multiple processes serving Web services. However, if the port number is modified in the application configuration file, then with the next Siebel Mobile Web Client instance a new siebel.exe process will start and listen to requests on the new port specified in the configuration file.

- Anonymous Web service requests are not supported.
- Chunked HTTP requests and responses are not supported.

Supported Authentication Formats

User authentication is the same as for the Siebel Mobile Web Client. The following authentication formats are supported:

- Username and password in the URL
- Username and password inside the SOAP header
- Username and password inside the Web Services-Security (WS-Security) header

Authentication Formats That Are Not Supported

The following authentication formats are not supported:

- Single sign-on (SSO)
- Stateful Web services using separate login and logout requests
- Stateless Web services using a session token

Enabling Web Services on the Siebel Mobile Web Client

Two new parameters have been added to the application configuration file to enable the Web service functionality: EnableWebServices and WebServicesPort.

To enable Web services on the Siebel Mobile Web Client

- Set the following parameters in the [Siebel] section of the application configuration file, such as uagent.cfg:

Parameter	Value
EnableWebServices	TRUE
WebServicesPort	Port number on which to listen. The default is 2330.

The next time the Siebel Mobile Web Client starts, it will start the siebel.exe child process. After the process has started, it can send requests and receive responses.

Starting the siebel.exe Process From the Command Line

When it is not required to start a Siebel Mobile Web Client instance, you can start the siebel.exe process independently using the command line.

To start the siebel.exe process from the command line

- Enter the following command:

```
 SIEBEL_CLIENT_ROOT\bin\siebel.exe /l <language_code> /c <configuration_file> /u
<username> /p <password> /d <datasource_in_cfg> /webservice <port_number>
```

For example:

```
C:\Siebel\client\bin\siebel.exe /l enu /c enu\uagent.cfg /u SADMIN /p SADMIN /d
Sample /webservice 2330
```

Note: To stop a siebel.exe process started from the command line, you must end the process from the Windows Task Manager.

Confirming that the siebel.exe Process is Listening

You can use the netstat utility from the DOS prompt to determine whether the siebel.exe child process is listening for Web service calls.

To confirm that the siebel.exe process is listening

1. From the DOS prompt, type the following:

   ```
   netstat -a -p TCP
   ```

2. Examine the output for the port number you set in the application configuration file, for example:

   ```
   TCP mycomputer:2330 mycomputer.mycompany.com:0 LISTENING
   ```

   `LISTENING` indicates that the siebel.exe process is listening for Web service calls.

Invoking Web Services on the Siebel Mobile Web Client

You can invoke Web services on the Siebel Mobile Web Client by passing credentials in the URL, in the SOAP header, or in the WS-Security header.

Example of Passing User Credentials in the URL

The URL format is:

```
http://<host>:<port>?SWEExtSource=WebService&Username=<username>
&Password=<password>
```

For example:

```
http://localhost:2330?SWEExtSource=WebService&Username=<username>
&Password=<password>
```

The following is an example of a request:

```
soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:asi="http://siebel.com/asi/">
 <soapenv:Header/>
 <soapenv:Body>
<asi:SiebelAccountQueryById>
 <PrimaryRowId>99-28B0A</PrimaryRowId>
 </asi:SiebelAccountQueryById>
 </soapenv:Body>
</soapenv:Envelope>
```

Example of Passing User Credentials in the SOAP Header

The URL format is:

```
http://<host>:<port>?SWEExtSource=WebService&WSSOAP=1
```

For example:

```
http://localhost:2330?SWEExtSource=WebService&WSSOAP=1
```

The following is an example of a request:

```
soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:asi="http://siebel.com/asi/">
 <soapenv:Header>
 <UsernameToken xmlns="http://siebel.com/webservices">SADMIN</UsernameToken>
```

```
 <PasswordText xmlns="http://siebel.com/webservices">SADMIN</PasswordText>
 <SessionType xmlns="http://siebel.com/webservices">None</SessionType>
 </soapenv:Header>
 <soapenv:Body>
<asi:SiebelAccountQueryById>
 <PrimaryRowId>99-28B0A</PrimaryRowId>
 </asi:SiebelAccountQueryById>
 </soapenv:Body>
</soapenv:Envelope>
```

Example of Passing User Credentials in the WS-Security Header

The URL format is:

```
http://<host>:<port>?SWEExtSource=SecureWebService&WSSOAP=1
```

For example:

```
http://localhost:2330?SWEExtSource=SecureWebService&WSSOAP=1
```

The following is an example of a 2002 request:

```
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soapenv="http://
schemas.xmlsoap.org/soap/envelope/" xmlns:asi="http://siebel.com/asi/">
 <soapenv:Header>
 <wsse:Security xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext">
 http://schemas.xmlsoap.org/ws/2002/07/secext
 <wsse:UsernameToken xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
 <wsse:Username>SADMIN</wsse:Username>
 <wsse:Password Type="wsse:PasswordText">SADMIN</wsse:Password>
 </wsse:UsernameToken>
 </wsse:Security>
 </soapenv:Header>
 <soapenv:Body>
<asi:SiebelContactQueryById soapenv:encodingStyle="http://schemas.xmlsoap.org/
soap/encoding/">
 <PrimaryRowId xsi:type="xsd:string">04-LLSQ5</PrimaryRowId>
 </asi:SiebelContactQueryById>
 </soapenv:Body>
</soapenv:Envelope>
```

The following is an example of a 2004 request:

```
<wsse:Security mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/wss/2004/
01/oasis-200401-wss-wssecuritysecext-1.0.xsd">
 <wsse:UsernameToken wsu:Id="UsernameToken-zsXRc97TujDINUug8ibD2Q22"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsswssecurity-
utility-1.0.xsd">
 <wsse:Username>SADMIN</wsse:Username>
 <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
usernametoken-profile-1.0#PasswordText">SADMIN</wsse:Password>
 <wsse:Nonce EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-soap-message-security-1.0#Base64Binary">f61vAYvDD0t2sUFEmXSVU+FlOvA=</
wsse:Nonce>
 <wsu:Created>2014-05-13T17:27:33Z</wsu:Created>
 </wsse:UsernameToken>
</wsse:Security>
```

Consuming External Web Services Using Siebel Web Services

An outbound Web service acts as a proxy to a Web service published by an external application. This process creates services that you can then use in a business process, virtual business component (VBC), run-time event, or any other mechanism within the Siebel application that can call a business service.

Consumption of external Web services is a two-step process:

- A WSDL file is imported using Siebel Tools.
- The consumed Web service is published for run-time clients to use.

Additional steps can involve defining VBCs based on the Web service.

The following outbound Web services topics are covered:

- *Creating an Outbound Web Service Based on a WSDL File*
- *Creating an Outbound Web Service Manually*
- *Integration Objects as Input Arguments to Outbound Web Services*
- *Web Services Support for Transport Headers*
- *Web Services Support for Transport Parameters*
- *SHA2 Support for Outbound Web Service*

Creating an Outbound Web Service Based on a WSDL File

Consumption of external Web services is accomplished using the WSDL Import Wizard in Siebel Tools. The procedure in this topic describes how to use this wizard to read an external WSDL document.

As of Siebel Innovation Pack 2014, you can import the following kinds of cyclic WSDL:

- Different namespace for same type and same element name
- Different element name for same type and same namespace
- Indirect cycle
- Direct cycle with same element name, type, and namespace

Note the following restrictions on WSDL import:

- The WSDL Import Wizard expects each schema to have a unique target namespace. Using the same namespace for more than one schema will generate an error.
- Importing a WSDL with a mix of different SOAP operation styles (for example, RPC and Document) within one service port binding is not supported. Modify the WSDL to have a service port binding defined for each SOAP operation style.

To create an outbound Web service based on a WSDL file

1. In Siebel Tools, create a new project and lock the project, or lock an existing project.
2. From the File menu, choose New Object to display the New Object Wizards dialog box.

3. Click the EAI tab, and then double-click Web Service.

   The WSDL Import Wizard appears.

   a. Select the project where you want the objects to be held after they are created from the WSDL document.
   b. Specify the WSDL document that contains the Web service or Web services definition that you want to import.
   c. Specify the file where you want to store the run-time data extracted from the WSDL document or accept the default.
   d. Specify the log file where you want errors, warnings, and other information related to the import process to be logged or accept the default.
   e. (Optional) Select the Process Fault Schema checkbox, and specify an existing Fault Integration Object Name, to create and reuse SOAP fault integration objects.

      > **Note:** SOAP fault integration objects are prepended with Fault_.

      For more information on SOAP fault integration objects, see *About SOAP Fault Schema Support*.

4. Click Next.

   A summary of your import information, as well as any errors, appears.

5. (Optional) Select the Deploy the Integration Object(s) and the Proxy Business Service(s) checkbox to deploy these objects to the Siebel run-time database.

   Deployed integration objects are shown in the Administration - Web Services screen, Deployed Integration Objects view in the Siebel client. Deployed business services are shown in the Administration - Business Services screen in the Siebel client.

   > **Note:** If you deploy integration objects while the Siebel Server is running, then you must subsequently clear the Web services cache in the Administration - Web Services screen, Inbound (or Outbound) Web Services view.

6. Click Finish to complete the process of importing the business service into the Siebel repository.

This procedure generates three objects in the Siebel repository:

- An outbound proxy business service of CSSWSOutboundDispatcher class. This service acts as a client-side implementation of the Web service and includes the operations and the arguments to the operations defined in the WSDL document.

  > **Note:** For RPC services, the order of input arguments is important. You can set the order through the Preferred Sequence property of the business service method argument in Siebel Tools. By specifying this parameter, the outbound dispatcher makes sure that the sequence parameters for an operation are in the correct order. The Preferred Sequence property is only supported with outbound services.

- Integration objects, representing input and output parameters of the service methods, if any of the operations require a complex argument (XML Schema) to be passed. If the service does not use complex arguments, then no integration object definitions will be created.
- A Web service administration document (XML file) containing the run-time Web service administration data to be imported into the Siebel Web Client, using the Outbound Web Services view of the Administration - Web Services screen.

  > **Note:** This is applicable only for the DR environment.

The purpose of the document is to allow administrators to modify run-time parameters such as the URL and encoding rules. The data contained within the document is used by the Web Services Dispatcher to assemble the SOAP document, to set any HTTP headers required (for example, soapAction), and to route the request to the correct URL. For information on how to migrate to runtime environment, see *Migrating Outbound Web Services*.

Migrating Outbound Web Services

You can migrate outbound web services to a run-time environment.

To migrate outbound web services, perform the following tasks:

1. Create migration rules with the following tables:
   - S_WS_WEBSERVICE
   - S_WS_OPERATION
   - S_WS_BNDNG_DTL
   - S_WS_PORT
   - S_WS_PORT_TYPE
   - S_WS_PORT_OPER

   For more information, see “Creating Migration Rules” in the Database Upgrade Guide.
2. Export the created rules to generate the datamig.inp and datamig.rul files.
3. Create a migration plan for export and import using Application Data Service. For more information, see “Creating a Migration Plan” in the Database Upgrade Guide.

> **Note:** You must select the Migration Application Data Service as a resource while creating a Migration plan. To migrate the outbound web services, you must execute this Migration Plan. For more information, see “Executing a Siebel Migration Plan” in the Database Upgrade Guide.

Creating an Outbound Web Service Manually

WSDL does not provide native bindings for EAI MQSeries and EAI MSMQ transports. If your business requires you to pick up messages using these transports, then you can manually create an outbound Web service definition and update a corresponding business service in Siebel Tools to point to that Web service. The following procedure describes this process.

To manually create a new outbound Web service

1. Navigate to the Administration - Web Services screen, Outbound Web Services view.
2. In the Outbound Web Services list applet, create a new record:
   a. Enter the namespace of the Web service in the Namespace column.
   b. Enter the name of the Web service in the Name column.
   c. Select Active or Inactive in the Status field.
   d. Enter a description of the Web service in the Comment column.

> **Note:** When importing an external Web service, you do not have to specify the proxy business service, integration objects, or the run-time parameters.

3. In the Service Ports list applet, create a new outbound service ports record:
   a. Enter the name of the Web service port in the Name column.
   b. Select a transport name for the protocol or queuing system for the Transport.
4. Enter the address for the transport chosen to publish the Web service:
   - The URL format to publish using HTTP is:

     ```
     http://webserver/eai_anon_lang/
     start.swe?SWEExtSource=SecureWebService&SWEExtCmd=Execute
     ```

     where:

     webserver is the name of computer where the Siebel Web Server is installed

     lang is the default language of the Object Manager that handles the request
   - The format to publish using the EAI JMS Transport is:

     ```
     jms://queue name@connection factory
     ```

     where:

     queue name is the Java Naming and Directory Interface (JNDI) name of the queue

     connection factory is the JNDI name of the JMS connection factory

     > **Note:** The JNDI name varies depending upon the JMS provider and your implementation.

     - The format to publish over the EAI MQSeries or EAI MSMQ transport is:

       ```
       mq://queue name@queue manager name

       msmq://queue name@queue machine name
       ```

       where:

       queue name is the name of the queue that is specified by either the EAI MQ Series or the EAI MSMQ transport at the time of its design

       queue manager name is the name of the EAI MQSeries Transport queue manager

       queue machine name is the name of the computer that owns the queue specified by the physical queue name for the EAI MSMQ Transport

       > **Note:** When publishing using EAI MQSeries or EAI MSMQ, you cannot generate WSDL files.

- For the Local Workflow Process or Local Business Service transport, enter the name of the workflow or business service to be called.
- For the Local Web Service transport, enter the name of the inbound port.

5. Select the binding that will publish the Web service.

   Note: RPC_Encoded, RPC_Literal, DOC_Literal, and Property Set styles of binding are supported for publishing Web services.

   Use the Property Set binding when the input property set to the proxy service is forwarded without changes to the destination address. This is intended primarily for use in combination with the Local Workflow Process or Local Business Service transport to avoid the overhead of processing XML.
6. Enter a description of the port in the Comment column.
7. In the Operations list applet, create a new operation record for the new service port you created in Step 3:
   a. Select the name of the business service method in the Method Display Name column to complete the process.
   b. Select the authentication type from the drop-down list.

      Note: For more information on using the Username/Password Authentication Type, see *About Web Services Security Support*.
8. Generate the WSDL file. For information, see *Generating a WSDL File*.

Updating the Outbound Proxy Business Service

When you have created your outbound Web service, update a corresponding outbound proxy business service in Siebel Tools to point to that Web service. This associates the outbound proxy business service and the outbound Web service. The following procedure outlines the steps you take to accomplish this task.

To update an outbound Web service proxy business service to point to an outbound Web service

1. In Siebel Tools, select the outbound Web service proxy business service you want to use to call your outbound Web service.
2. Add the following user properties for this business service and set their values based on the outbound service port of your Web service:
   - siebel_port_name
   - siebel_web_service_name
   - siebel_web_service_namespace

Integration Objects as Input Arguments to Outbound Web Services

It is recommended that the property set used as an input argument to the outbound Web service have the same name as the input argument's outbound Web service proxy.

You can do this using one of the following options:

- Change the output from all your business services that provide the input to the outbound Web service from SiebelMessage to the actual outbound Web service argument name specified in Siebel Tools.

Change the output from your business services in Siebel Tools, as well as the name of the property set child that contains the integration object instance.

- Change the property set type name from SiebelMessage to the actual outbound Web service argument name by using Siebel eScript on a business service before calling the outbound Web service.

The following Siebel eScript example shows how to pass an integration object and a session token to a proxy business service using the integration object as an input argument. The script is written on the Service_PreInvokeMethod event of the proxy business service.

```
function Service_PreInvokeMethod (MethodName, Inputs, Outputs) {
var childPS;
var newInputPS;
var svc;

for (var i = 0; i < Inputs.GetChildCount(); i++) {
 if(Inputs.GetChild(i).GetType() == "SiebelMessage") {
childPS = Inputs.GetChild(i);
 }
}
childPS.SetType("myBusSvcMethod:myIntegrationObject");
newInputPS = TheApplication().NewPropertySet();
newInputPS.SetProperty("myBusSvcMethod:sessionToken:string",
Inputs.GetProperty("token"));
newInputPS.AddChild(childPS);

svc = TheApplication().GetService("myBusSvc");
svc.InvokeMethod("myBusSvcMethod", newInputPS, Outputs);
return (CancelOperation); // must use CancelOperation with custom methods

}
```

Web Services Support for Transport Headers

The outbound Web service dispatcher supports input arguments for user-defined (or standard) transport headers.

The following is the format for the outbound Web service dispatcher input arguments:

- Name: siebel_transport_header:headerName
- Value: Header value

The following table shows examples of input arguments for transport headers.

Parameter Name	Value
siebel_transport_ header:UserDefinedHeader	myData
siebel_transport_header:Authorization	0135DFDJKLJ

Web Services Support for Transport Parameters

The outbound Web service dispatcher supports input arguments for transport parameters defined in proxy business services such as EAI HTTP Transport.

The following is the format for the outbound Web service dispatcher input arguments:

- Name: siebel_transport_param:parameterName
- Value: Parameter value

The following table shows examples of input arguments for transport parameters.

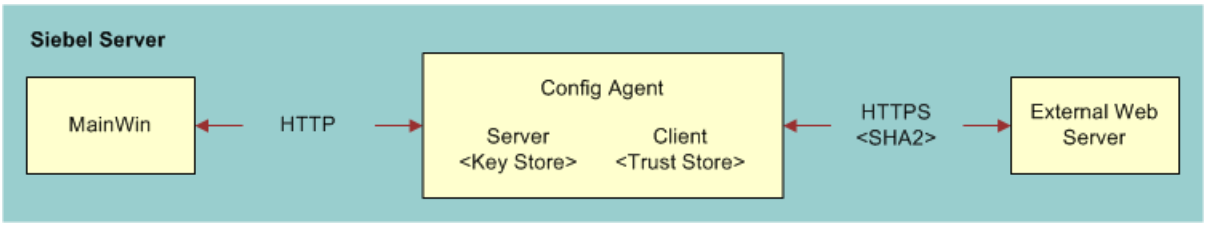
Parameter Name	Value
siebel_transport_ param:HTTPRequestMethod	HTTP method to use with the data request, such as Post or Get
siebel_transport_ param:HTTPRequestURLTemplate	Template for the request URL, which is the address to which the data is sent or from which a response is requested, for example: `http://mycompany.com/*`

For more information on transport parameters, see the topic on EAI HTTP Transport business service method arguments in *Transports and Interfaces: Siebel Enterprise Application Integration* .

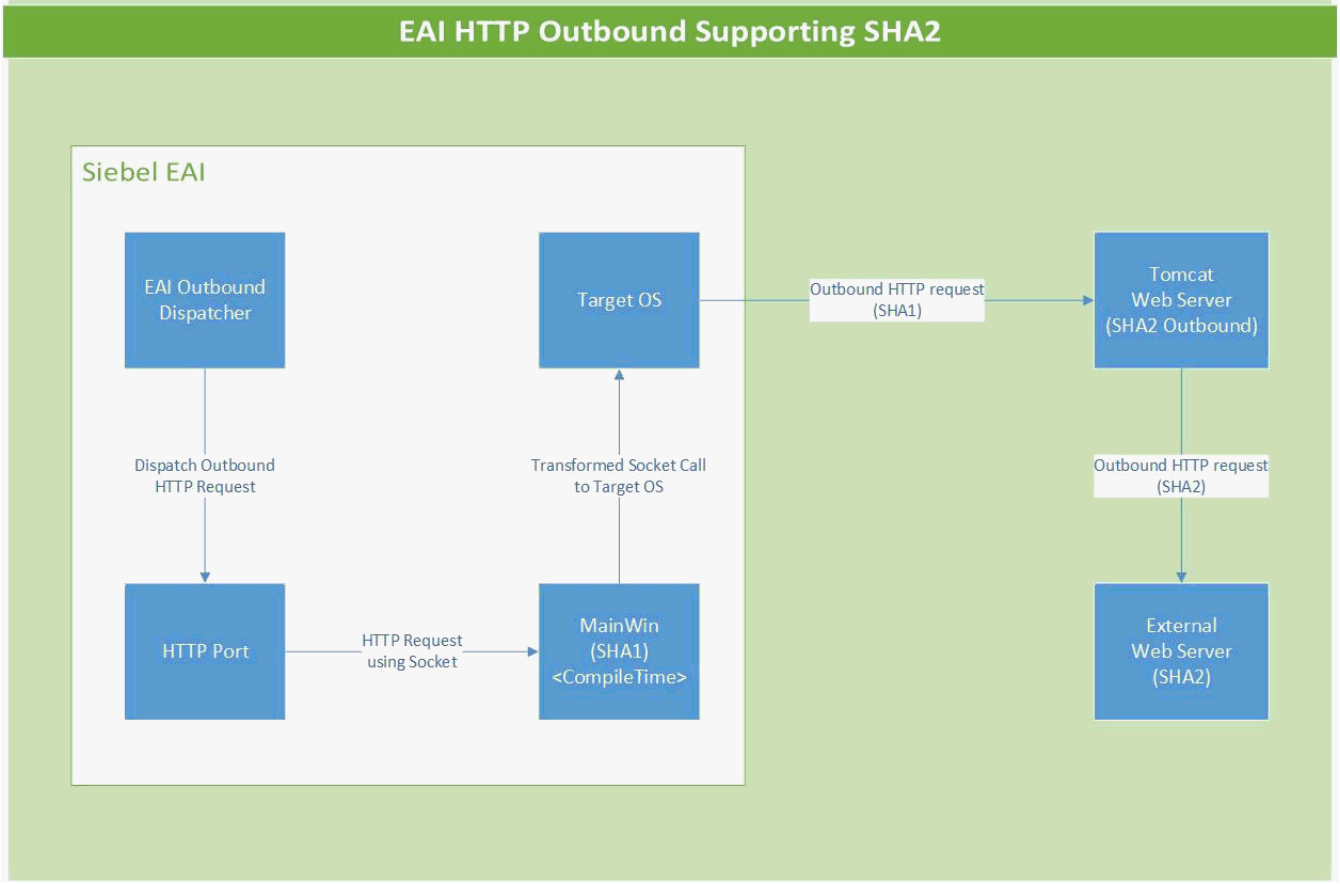
SHA2 Support for Outbound Web Service

> **Note:** Non-Windows platforms use the Mainwin library, which does not support SHA2. In order to provide SHA2 support on non-Windows platforms, Siebel CRM uses the OUTBOUNDSHA2 named subsystem, which routes the outbound calls using a javacontainer. Because Microsoft Windows supports SHA2, Siebel CRM applications on Windows do not require OUTBOUNDSHA2.

Siebel CRM supports SHA2 for outbound calls through the framework described here. This support for SHA2 is through the introduction of a Config Agent between the Siebel Server and the external Web Server. The Config Agent accepts local requests from Mainwin within the Siebel Server and transfers the same to the external Web Server in SHA2, as seen in the following figure. To configure certificates for the Config Agent, see the chapter on communications and data encryption in the *Siebel Security Guide* .



The transfer is made possible by a servlet named outboundeai that resides on the Config Agent. This servlet copies the outbound request body and Siebel Server header information and transfers it to the external Web Server. The servlet also collects the response from the Web Server and transfers it back to the Siebel Server, as seen in the following figure.



Configuring Siebel Server and Config Agent for SHA2 Outbound

SHA2 support for HTTP outbound is achieved via configuring a named subsystem of type JavaContainerSubSys. The name of this named subsystem is then set to the value of the EAIOutboundSubSys component parameter, as described in the following procedure.

To configure the named subsystem for SHA2 support in outbound

1. Go to the Siebel Server Manager and search on OUTBOUNDSHA2 as follows:

   ```
   list param for named subsystem OUTBOUNDSHA2
   ```

   where *OUTBOUNDSHA2* is the name of the new named subsytem.
2. Set the name of this named subsystem as the value of the EAIOutboundSubSys component parameter:

   ```
   change param EAIOutboundSubSys=OUTBOUNDSHA2 for comp SCCObjMgr_enu
   ```

   where *SCCObjMgr_enu* is the Siebel Object Manager component.

> **Note:** If the object manager component you are trying to use does not support the EAIOutboundSubSys component parameter, then send the EAIOutboundSubSys component parameter to the EAI HTTP Transport or outbound proxy business service method.

- To send the EAIOutboundSubSys component parameter to any method of the EAI HTTP Transport business service:

  ```
  EAIOutBoundSubSys=OUTBOUNDSHA2
  ```

- To send the EAIOutboundSubSys component parameter to any method of the outbound proxy business service:

  ```
  siebel_transport_param:EAIOutBoundSubSys=OUTBOUNDSHA2
  ```

3. When the user makes an outbound call, the EAI Outbound Dispatcher checks for the value of the component parameter.
   a. If the value is present, the dispatcher will make a http call to the servlet hosted in the Config Agent specified in CONTAINERURL and the Config Agent will make the https call to the external Web Server.
   b. If the value is not present for the component parameter or the named subsystem parameter, the dispatcher makes a direct HTTPS call to the external Web Server.
4. Restart the component.

   ```
   stop comp sccobjmgr_enu
   start comp sccobjmgr_enu
   ```

Parameters for the named subsystem

The named subsystem has three parameters as seen in the following table.

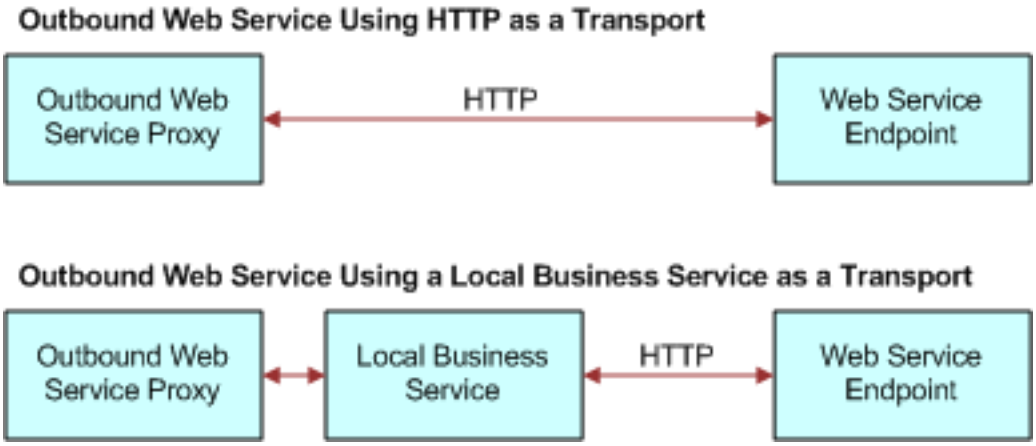
> **Note:** Only CONTAINERURL should be used for this configuration.

Property	Value
CLASSPATH	Not applicable.
OPTIONS	Not applicable.
CONTAINERURL	URL for siebel-eaioutbound.war hosted on the Config Agent.

Using the Local Business Service

In many instances, Web services use specialized SOAP headers for common tasks such as authentication, authorization, and logging. To support this common Web service extensibility mechanism, the Local Business Service transport for outbound Web services can be used. When specified as a transport, the Web services infrastructure will route the

message to the specified business service for additional processing and delivery to the Web service endpoint as shown in the first half of the following figure.



If the Web service to be invoked is within the sample application, then no need exists to call such a Web service by using HTTP (or anything else).

An example of using a local business service is a department store developing a workflow in Siebel Tools to perform credit card checks before purchases. The purchase is entered into the sales register along with the credit card information (the outbound Web service proxy). If the credit card is issued by the department store, then the information can be checked using the internal database (a local business service). The send request stays within the department store's own computer network. An approval or denial is the output (the Web service endpoint). If the credit card is a MasterCard or a Visa card, then the card information is passed over the Internet for verification. No local business service would be involved.

The input to the local business service is a property set representation of the SOAP request. Once within the local business service, additional SOAP headers can be added to address infrastructure requirements by direct modification of the input property set by using Siebel eScript or Siebel VB.

The following local business service topics are also discussed:

- *Script Example for a Local Business Service*
- *SOAP Document Generated by the Local Business Service*
- *Using the Local Business Service in an Outbound Web Service*

Script Example for a Local Business Service

A portion of the sample script for a local business service used to add a custom SOAP header to an outbound Web service request is shown in the following example. Local variables, error handling, and object destruction are omitted for clarity.

```
// Create the SOAP header.
soapHdr.SetType("SOAP-ENV:header");

// Populate the SOAP header elements.
appId.SetType("ns1:ApplicationID");
appId.SetValue("Siebel");
pwd.SetType("ns1:PWS");
pwd.SetValue("123456789");
langCd.SetType("ns1:Lang");
langCd.SetValue("ENU");
uName.SetType("ns1:userID");
uName.SetValue("first.last@mycompany.com");
```

```
// Populate the profileHeader element.
profileHeader.SetType("authHeader");
profileHeader.SetProperty("xmlns:ns1", "http://siebel.com/authHeaders");
profileHeader.AddChild(appId);
profileHeader.AddChild(pwd);
profileHeader.AddChild(langCd);
profileHeader.AddChild(uName);

// SOAP header property set. Once this is complete, add the SOAP header as a child
of the Input property set (which contains the SOAP:body).

soapHdr.InsertChildAt(profileHeader, 0)

Inputs.InsertChildAt(soapHdr, 0);

// Convert the property set to a well-defined XML document.
// Using the XML Hierarchy Converter: must add a child element of type XMLHierarchy.
childPS.SetType("XMLHierarchy");
childPS.AddChild(Inputs);
inPs.AddChild(childPS);
inPs.SetProperty("EscapeNames", "FALSE");
inPs.SetProperty("GenerateProcessingInstructions", "FALSE");
xmlSvc.InvokeMethod("XMLHierToXMLDoc", inPs, outPs);

// Proxy the request through a trace utility to view the SOAP document.
// Set custom HTTP header - SOAPAction
outPs.SetProperty("HTTPRequestURLTemplate", "http://localhost:9000/search/beta2");
outPs.SetProperty("HTTPRequestMethod", "POST");
outPs.SetProperty("HTTPContentType", "text/xml; charset=UTF-8");
outPs.SetProperty("HDR.SOAPAction","customSOAPActionValue");

// Invoke the Web service using the standard HTTP protocol.
httpSvc.InvokeMethod("SendReceive", outPs, hpOut);
// Convert the SOAP document to a property set using the XML Converter, returning
the SOAP header and SOAP body.
xmlCtr.InvokeMethod("XMLToPropSet", hpOut, Outputs);
…
```

After you have created your business service, deliver its workspace.

SOAP Document Generated by the Local Business Service

The following example displays the resulting SOAP document generated by the *Script Example for a Local Business Service*. The addition of the authHeader element to the SOAP header corresponds to the structure defined in the sample code sections that populate the SOAP header and profileHeader elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:header>
 <authHeader xmlns:ns1="http://siebel.com/authHeaders">
 <ns1:ApplicationID>Siebel</ns1:ApplicationID>
 <ns1:PWS>123456789</ns1:PWS>
 <ns1:Lang>ENU</ns1:Lang>
 <ns1:userID>first.last@mycompany.com</ns1:userID>
 </authHeader>
 </SOAP-ENV:header>
 <SOAP-ENV:Body>
 …
```

```
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Using the Local Business Service in an Outbound Web Service

You use the Outbound Web Services view in the Administration - Web Services screen to configure an outbound Web service to use the local business service created by *Script Example for a Local Business Service*.

To use the local business service in an outbound Web service

1. In the Siebel client, navigate to the Administration - Web Services screen, Outbound Web Services view.
2. In the Outbound Web Services list, select the desired outbound Web service.
3. In the Service Ports list, set the following properties:

Name	Value
Transport	Local Business Service
Address	Name of the local business service

4. Restart the Siebel Server component to allow the changes to take effect.

About XML Schema Support for the xsd:any Tag

In the current framework, WSDL Import Wizard makes use of the XML Schema Import Wizard to create integration objects to represent hierarchical data. Integration objects are meant to be strongly typed in the Siebel application. You are now able to import a schema that uses the xsd:any tag, which indicates a weakly typed data representation, and to possibly create an integration object from it.

Mapping the xsd:any Tag in the WSDL Import Wizard

In the WSDL Import Wizard, two possible mappings exist for the xsd:any tag. The tag can be mapped as an integration component or as an XMLHierarchy on the business service method argument.

The xsd:any tag can contain a *namespace* attribute. If the value for that attribute is known, then one or more integration components or even an integration object can be created. If the value for that attribute is not known, then the business service method argument for that particular wsdl:part tag is changed to data type Hierarchy, consequently losing any type information.

The value for the attribute being known refers to the following situations:

- A schema of *targetNamespace* value, being the same as that of the namespace attribute value, is imported by way of the xsd:import tag.
- A schema of *targetNamespace* value, being the same as that of the namespace attribute value, is a child of the wsdl:types tag.

For the case of being known, all the global elements belonging to the particular schema of that targetNamespace will be added in place of the tag. One or more integration components can potentially be created.

Another tag similar to the xsd:any tag is the xsd:anyAttribute tag. The mapping is similar to that of the xsd:any tag. In this case, one or more integration component fields can be created.

The xsd:anyAttribute tag has a *namespace* attribute. If the namespace value is known (the conditions for being known were previously noted in this topic), then all the global attributes for that particular schema will be added in place of this tag. Therefore, one or more integration component fields can potentially be created.

In the case where the namespace value is not known, then the wsdl:part tag that is referring to the schema element and type will be created as data type Hierarchy.

Mapping the xsd:any Tag in the XML Schema Wizard

For the case of the XML Schema Wizard, there is only one possible mapping for the xsd:any tag, namely as an integration component.

The xsd:any tag can contain a *namespace* attribute. If the value for that attribute is known, then one or more integration components or even an integration object can be created. If the value for that attribute is not known, then an error will be returned to the user saying that the integration object cannot be created for a weakly typed schema.

The value for the attribute being known refers to the situation of the XML Schema Wizard where a schema of targetNamespace value, being the same as that of the namespace value, has been imported by way of the xsd:import tag.

For the case of being known, all the global elements belonging to the particular schema of that targetNamespace will be added in place of the tag. So, one or more integration components can potentially be created.

The mapping of the xsd:anyAttribute is similar to that of the xsd:any tag. In this case, one or more integration component fields can be created.

The xsd:anyAttribute tag has a *namespace* attribute. If the namespace value is known (the condition for being known was previously noted in this topic), then all the global attributes for that particular schema will be added in place of this tag. Therefore, one or more integration component fields can potentially be created.

In the case where the namespace value is not known, then an error is returned to the user stating that an integration object cannot be created for a weakly typed schema.

Examples of Invoking Web Services

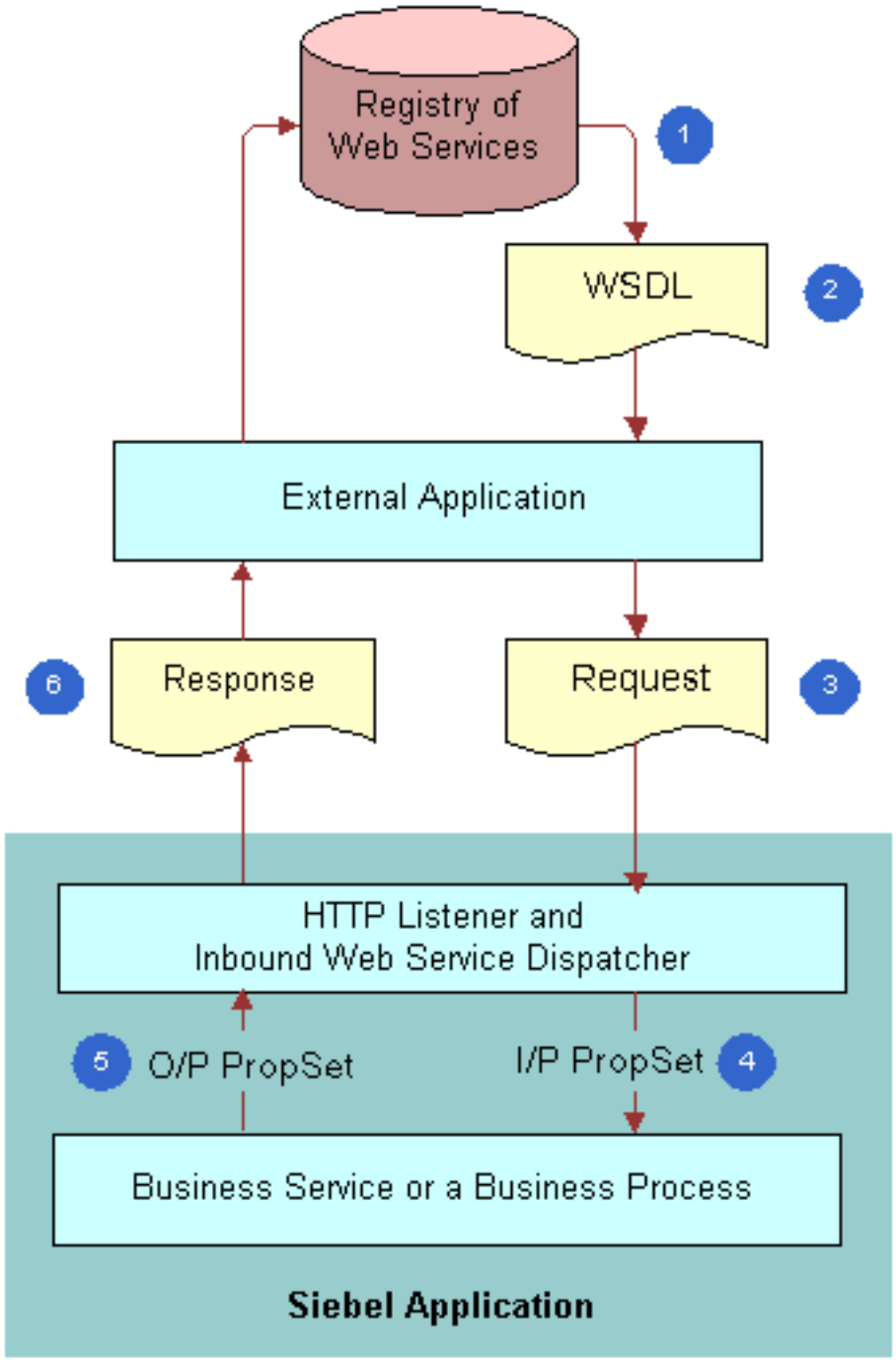
The following two examples show sample flows of how to call an external Web service from a Siebel application, or how to call a Siebel Web service from an external application.

Invoking an External Web Service Using Workflow or Scripting

As illustrated in the following figure, the following steps are executed to call an external Web service:

1. The developer obtains the Web service description as a WSDL file.
2. The WSDL Import Wizard is called.
3. The WSDL Import Wizard generates definitions for outbound proxy, integration objects for complex parts, and administration entries.
4. The Outbound Web Service proxy is called with the request property set.

5. The request is converted to an outbound SOAP request and sent to the external application.
6. The external application returns a SOAP response.
7. The SOAP response is converted to a property set that can be processed by the caller, for example, Calling Function.



8. The following example shows how to invoke Web services using Siebel eScript:

```
function Service_PreCanInvokeMethod (MethodName, &CanInvoke) {
```

```
if (MethodName == "invoke") {

CanInvoke = "TRUE";

return (CancelOperation);

}

else

return (ContinueOperation);

}

function Service_PreInvokeMethod (MethodName, Inputs, Outputs) {

if (MethodName == "invoke") {

var svc = TheApplication().GetService("CustomerDBClientSimpleSoap");

var wsInput = TheApplication().NewPropertySet();

var wsOutput = TheApplication().NewPropertySet();

var getCustInput = TheApplication().NewPropertySet();

var listOfGetCustomerName = TheApplication().NewPropertySet();

var getCustomerName = TheApplication().NewPropertySet();

try {

// obtain the customer ID to query on. This value will be provided in the input property set

var custId = Inputs.GetProperty("custId");

// set property to query for a customer ID with a value of '1'

getCustomerName.SetType("getCustomerName");

getCustomerName.SetProperty("custid", custId);

// set Type for listOfGetCustomerName
```

```
listOfGetCustomerName.SetType("ListOfgetCustomerName");

// set Type for getCustInput

getCustInput.SetType("getCustomerNameSoapIn:parameters");

// assemble input property set for the service.

listOfGetCustomerName.AddChild(getCustomerName);

getCustInput.AddChild(listOfGetCustomerName);

wsInput.AddChild(getCustInput);

// invoke the getCustomerName operation

svc.InvokeMethod("getCustomerName", wsInput, wsOutput);

// parse the output to obtain the customer full name check the type element on each PropertySet
(parent/child) to make sure we are at the element to obtain the customer name

if (wsOutput.GetChildCount() > 0) {

 var getCustOutput = wsOutput.GetChild(0);

 if (getCustOutput.GetType() == "getCustomerNameSoapOut:parameters") {

 if (getCustOutput.GetChildCount() > 0) {

 var outputListOfNames = getCustOutput.GetChild(0);

 if (outputListOfNames.GetType() == "ListOfgetCustomerNameResponse") {

 if (outputListOfNames.GetChildCount() > 0) {

 var outputCustName = outputListOfNames.GetChild(0);

 if (outputCustName.GetType() == "getCustomerNameResponse") {

 var custName = outputCustName.GetProperty("getCustomerNameResult");

 Outputs.SetProperty("customerName", custName);

 }
```

```
 }

 }

 }

 }

}

return (CancelOperation);

}

catch (e) {

TheApplication().RaiseErrorText(e);

return (CancelOperation);

}

}

else

return (ContinueOperation);

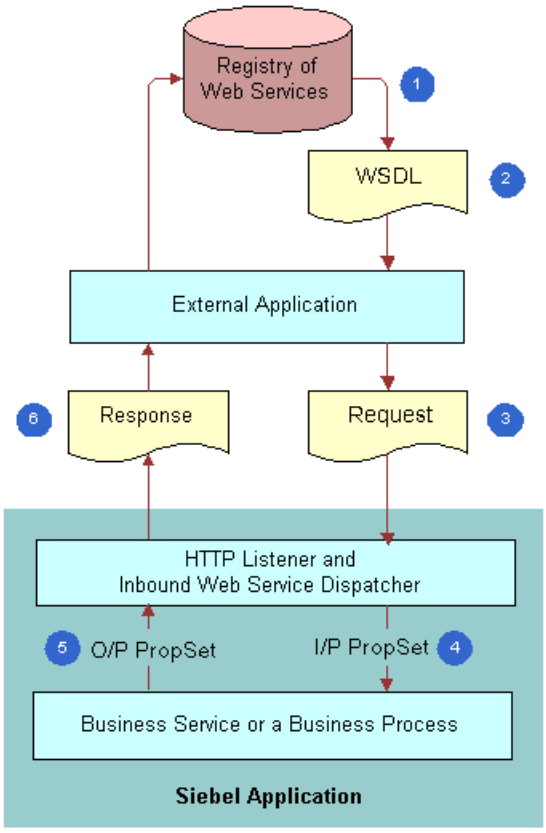
}
```

Invoking a Siebel Web Service from an External Application

As illustrated in the following figure, the following steps are executed to invoke a Siebel Web service from an external application:

1. The WSDL document for an active Web service is published in the Siebel Inbound Web Services view. To allow processing of the Web service requests, the developer has to make sure:
    a. The Web Server and the Siebel Server are up and running.
    b. The appropriate setup is done in the Siebel Server.
2. In the external application, the WSDL document is imported to create a proxy that can be used to invoke the Siebel Web service from Step 1.
3. The external application sends the SOAP request into the Siebel application.

4. The Web Service Inbound Dispatcher converts the SOAP request to a property set. Depending on the inbound Web service configuration, the property set is passed to a business service or a business process.
5. The property set is returned from the business service or business process to the Web Service Inbound Dispatcher.
6. Response is converted to a SOAP message and sent back to the invoking external application.



The following is an example of invoking a Siebel-published Web service using .NET.

```
// Removed using declaration
namespace sieOppClnt {
public class sieOppClnt : System.Web.Services.WebService {
public siebOptyClnt() {
InitializeComponent();
}
// WEB SERVICE CLIENT EXAMPLE
/* The optyQBE returns a list of opty based upon the required input params. Because
the input to the Siebelopty.QueryByExample method uses an Input/Output param,
ListOfInterOptyIntfaceTopElmt will be passed by ref to Siebel. To add the Siebel
Opportunity Web Service definition to the project, the wsdl.exe utility was run
to generate the necessary helper C# class for the service definition. */
[WebMethod]
public ListOfInterOptyIntfaceTopElmt optyQBE(string acctName, string acctLoc,
string salesStage) {
 Siebelopty svc = new Siebelopty();
 ListOfInterOptyIntfaceTopElmt siebelMessage = new
 ListOfInterOptyIntfaceTopElmt();
 ListOfInteroptyInterface optyList = new ListOfInteroptyInterface();
 opty[] opty = new opty[1];
 opty[0] = new opty();
 opty[0].Account = acctName;
 opty[0].AccountLocation = acctLoc;
 opty[0].SalesStage = salesStage;
```

```
/* Assemble input to be provided to the Siebel Web service. For the sake of
simplicity the client will query on the Account Name, Location, and Sales
Stage. Ideally, also check to make sure that correct data is entered. */

optyList.opty = opty;
siebelMessage.ListOfInteroptyInterface = optyList;

// Invoke the QBE method of the Siebel Opportunity business service
svc.SiebeloptyQBE(ref siebelMessage);

/* Return the raw XML of the result set returned by Siebel. Additional
processing could be done to parse the response. */

return siebelMessage;
}
}
}
```

About Web Services Security Support

Oracle endorses the industry standard known as the Web Services Security (WS-Security) specification. The WS-Security specification is a Web services standard that supports, integrates, and unifies multiple security models and technologies, allowing a variety of systems to interoperate in a platform- and language-independent environment.

By conforming to industry standard Web service and security specifications, secure cross-enterprise business processes is supported. You can deploy standards-based technology solutions to solve specific business integration problems.

For security support, you can also apply access control to business services and workflows. For more information on configuring access control, see *Siebel Security Guide* .

Configuring the Siebel Application to Use theWS-Security Specification

To use the WS-Security specification in the Siebel application, two parameters, UseAnonPool and Impersonate, must be set. An example of configuring WS-Security for Siebel inbound Web services follows.

To configure the Siebel application to use the WS-Security specification

1. Check Configure Anonymous Pool parameter in the basic information section of the eai_anon application in the Application Interface profile.

   For more information about configuring the anonymous pool, see *Siebel Performance Tuning Guide* .
2. Start the Siebel Server.
3. Navigate to the Administration - Server Configuration screen, Enterprises view, and then Profile Configuration.
4. In the Profile Configuration list, query the Alias field for SecureWebService.
5. Make sure that the SecureWebService profile (named subsystem) has parameters with the following values:

Parameter	Alias	Value
Service Method to Execute	DispatchMethod	Dispatch

Parameter	Alias	Value
Service to Execute	DispatchService	Web Service Inbound Dispatcher
Impersonate	Impersonate	True

6. When the client makes a call to the Web service, make sure that SWEExtSource points to the correct application name and named subsystem, for example:

```
http://myserver/siebel/app/eai_anon/enu/?SWEExtCmd=Execute
&SWEExtSource=SecureWebService
```

About WS-Security UserName Token Profile Support

Siebel CRM supports the WS-Security UserName token mechanism, which allows for the sending and receiving of user credentials in a standards-compliant manner. The UserName token is a mechanism for providing credentials to a Web service where the credentials consist of the UserName and Password. The password must be passed in clear text. The UserName token mechanism provides a Web service with the ability to operate without having the username and password in its URL or having to pass a session cookie with the HTTP request.

Note: Using WS-Security is optional. If it is critical that the password not be provided in clear text, then use HTTPS.

The following is an example of a UserName token showing the username and password:

```
<wsse:Security xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext">
 <wsse:UsernameToken xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
 <wsse:Username>WKANDINSKY</wsse:Username>
 <wsse:Password Type="wsse:PasswordText">AbstractArt123</wsse:Password>
 </wsse:UsernameToken>
</wsse:Security>
```

Note: If you are using Web single sign-on (SSO), then use the Siebel trust token value in wsse:Password instead of the password.

About Support for the UserName Token Mechanism

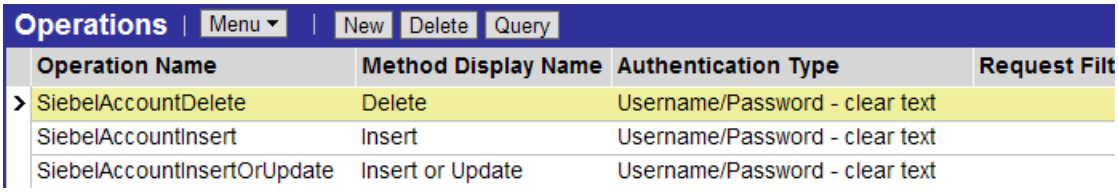
Support for the UserName Token mechanism includes the following:

- Allows an inbound SOAP request to contain user credentials that can be provided to the inbound SOAP dispatcher to perform the necessary authentication
- Allows an inbound SOAP dispatcher to perform the necessary authentication on an inbound SOAP request that contains user credentials
- Allows an outbound SOAP request to contain user credentials that can be utilized by the external application

Note: Passing user credentials in the URL is not supported in the current release of Siebel CRM.

Using the User Name Token for Inbound Web Services

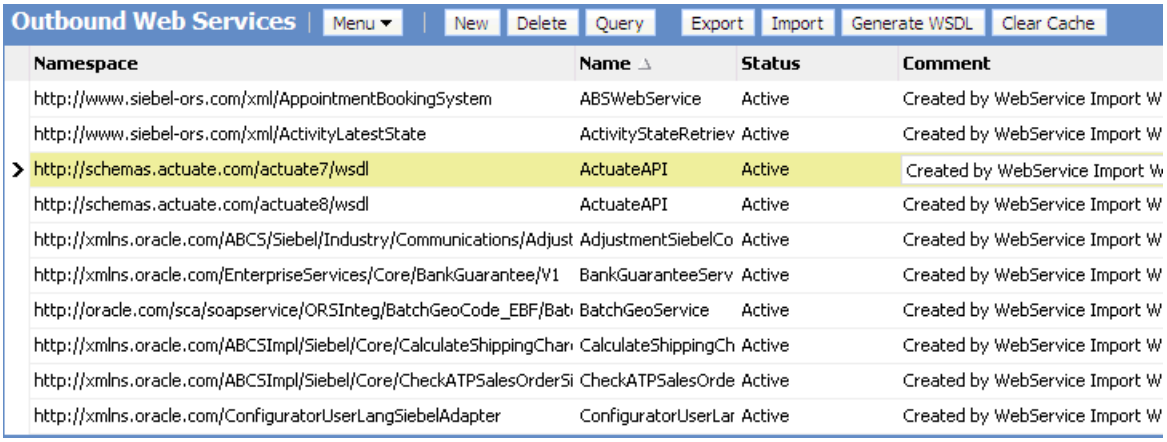
The Inbound Web Services view provides an interface for associating operations with authentication types. The names of the operations must be globally unique. The applet shown in the following figure can be defined as requiring a UserName Token with username and password provided in clear text.



Note: If you want to use Siebel Authentication and Session Management SOAP headers, then set the authentication type to None. For more information, see *About Siebel Authentication and Session Management SOAP Headers*.

Using the UserName Token for Outbound Web Services

Each Web service operation in the Outbound Web Services list applet might be tied to an authentication type by selecting from the Authentication Type picklist (see the following figure) in the Operations picklist, in the following applet.



Proxy Configuration for Java Web Container

If your enterprise network is connected to the Internet through a proxy Internet server then configure the Java Web Container to route its traffic through the enterprise proxy server. To do so, apply the following proxy configuration:

HTTP proxy configuration

In the `<installation_root>\ses\siebsrvr\javacontainer\javacontainer1\bin\setenv.bat` file, add after line:

```
set CATALINA_OPTS=-Dhttp.proxyHost=<proxy_server_name> -Dhttp.proxyPort=<port_num>
```

HTTPS proxy configuration:

In the `<installation_root>\ses\siebsrvr\javacontainer\javacontainer1\bin\setenv.bat` file, add after line:

```
set CATALINA_OPTS=-Dhttps.proxyHost=<proxy_server_name> -Dhttps.proxyPort=<port_num>
```

To configure MainWin and import certificates in to it, see the chapter on communications and data encryption in the *Siebel Security Guide* .

About Siebel Authentication and Session Management SOAP Headers

You can use Siebel Authentication and Session Management SOAP headers to send and receive user credentials and session information. You can send a username and password for login that calls one of the following sessions:

- One that closes after the outbound response is sent.
- One that remains open after the response is sent.

For example, a custom Web application can send a request that includes a username and password, and calls a stateless session, one that remains open after the outbound response is sent. The Siebel Server generates an encrypted session token that contains user credentials and a session ID. The Siebel Server includes the session token in the SOAP header of the outbound response. The client application is responsible for capturing the returned session token and including it in the SOAP header of the next request.

The Session Manager on the Application Interface (AI) extracts the user credentials and session ID from the session token and reconnects to the session on the Siebel Server. If the original session has been closed, then a new session is created.

You can use the SOAP headers listed in the following table to call different types of sessions and pass authentication credentials:

Note: The values entered are case insensitive.

SOAP Header Block	Description
SessionType	You use the SessionType SOAP header to define the type of session. Valid values are None, Stateless, Stateful, and ServerDetermine: • **None.** A new session is opened for each request and then closed after a response is sent out. The SessionType none might or might not include UsernameToken and PasswordText SOAP headers.

SOAP Header Block	Description
	When UsernameToken and PasswordText SOAP headers are included, these credentials are used for authentication. If the UsernameToken and PasswordText SOAP headers are excluded from the SOAP header, then anonymous login is assumed. The anonymous login requires additional configuration in the Siebel Application Interface profile and Named Subsystem configuration (AllowAnonymous equals (=) True, Impersonate equals (=) False). For more information about configuring anonymous login, see *Siebel Security Guide* . • **Stateless.** A new session is opened for an initial request and the session remains open for subsequent requests. Relogin occurs automatically (transparent to the user) if the session is closed. UsernameToken and PasswordText must be included as SOAP headers in the initial request to open a stateless session. Stateless session management is the best method to use for high-load Web service applications. Using Stateless mode, the application provides the username and password only once, that is for the initial request. A session is opened on the server and is dedicated for this user. In the response Siebel CRM returns the SessionToken, which is an encrypted string containing the information about username, password, and timestamp. For subsequent requests the application must use the SessionToken to reuse the session. For security reasons, SessionTokens are returned for each response. The application must provide the last received SessionToken for the next request. The SessionToken-Siebel session map is maintained in the Siebel Application Interface (AI); based on the SessionToken value, AI sends the request to the correct Siebel session (task). Although the session is persistent, authentication happens for each request (AI decrypts the UserName and Password from the SessionToken). • **Stateful.** A new, dedicated session is opened for an initial request and the session remains open for subsequent requests. Relogin does not occur automatically if the session is closed. UsernameToken and PasswordText must be included as SOAP headers in the initial request to open a stateful session. As with Stateless sessions, Siebel CRM returns the SessionToken in the response. For subsequent requests the application must use the SessionToken to reuse the session. Unlike Stateless sessions, transparent failover (automatic relogin) is not supported. This is because Stateful sessions might have state information stored that makes it mandatory to connect to the same task for each request. • **ServerDetermine.** A new session is established to Siebel, and a series of subsequent requests is served. The Siebel Server is free to multiplex the session to serve other users if possible, but the client is free to make stateful calls to Siebel. Failover is not supported for this mode. ServerDetermine provides the most flexibility: the session can be dedicated or not. If the number of users increases and resources must be recovered, then the session state is written to the database so that it can be restored. The session can then serve other users. If SessionType is absent, then the default value is None, and the session will be closed after the request is processed.
UsernameToken	You use the UsernameToken SOAP header to send the Login ID to the Siebel Server.
PasswordText	You use the PasswordText SOAP header to send the password used by the login ID to the Siebel server. If using Web single sign-on (SSO), then use the Siebel trust token value in PasswordText instead of the password.

SOAP Header Block	Description
SessionToken	Session tokens are used with stateless requests. They are sent and received using the SessionToken SOAP header. After receiving an initial request with valid authentication credentials and a session type set to Stateless, the Siebel Server generates a session token and includes it in the SOAP header of the outbound response. The session token is encrypted and consists of a session ID and user credentials. The custom Web application uses the session token for subsequent requests. The Session Manager on the AI extracts a session ID and user credentials from the session token, and then passes the information to the Siebel Server. The session ID is used to reconnect to an existing session or automatically log in again if the session has been terminated. **Note:** Reconnecting or automatic logging in again will only happen if the token has not timed out. If it times out, then the user must manually log in again. Token timeout must be greater than or equal to session timeout. For more information on session token timeout, see *Session and Session Token Timeout-Related Parameters*. However, the session token must be changed to the new one sent on every response. The session token has a maximum time to live, which can invalidate it even if its timeout (for being inactive) has not been reached. Always get the newest session token returned by the response and use it on the next request. The same session token must not be used by concurrent requests, because having multiple requests point to the same session can cause errors.

For examples of using SOAP headers for session management and authentication, see *Examples of Using SOAP Headers for Authentication and Session Management*.

The namespace used with Siebel Authentication and Session Management SOAP headers is:

```
xmlns="http://siebel.com/webservices"
```

Note: The Siebel Session Management and Authentication SOAP headers are different from the SOAP headers used for WS-Security. Do not use the two types of header together.

Combinations of Session Types and Authentication Types

The following table summarizes the combinations of authentication types and session types.

Authentication Type	Session Type	Description
None	None	A single request is sent with an anonymous user login, and the session is closed after the response is sent out. In order for the anonymous session to be identified by the AI Plug-in, UsernameToken and PasswordText must be excluded in the SOAP headers.
Username and password	None	A single request is sent with the username and password used to log in, and the session is closed after the response is sent out.

Authentication Type	Session Type	Description
Username and password	Stateless	The initial request to log in establishes a session that is to remain open and available for subsequent requests. Username/password are used to log in and a session token is returned in a SOAP header included in the outbound response. The session remains open.
Session token (stateless)	Stateless	Request to reconnect to an established session, using the information contained in the session token. If the session has been closed, then automatic relogin occurs. The Siebel servers include the session token in the SOAP header of the response. The session remains open.
Session token (stateless)	None	When a SOAP header carries a session token and has the session type set to None, then the Session Manager on the AI closes (logs out) of this session, and invalidates the session token. The session token is not used after the session is invalidated.

For examples that illustrate some of these combinations, see *Examples of Using SOAP Headers for Authentication and Session Management*.

Enabling Session Management on AI

To enable Session Management on the Application Interface (AI) for SOAP header handling, the Web service request must include the following URL parameter: WSSOAP=1. For example:

```
http://mywebserver/siebel/app/eai/enu/
swe?SWEExtSource=CustomUI&SWEExtCmd=Execute&WSSOAP=1
```

Note: When using Siebel Session Management and Authentication SOAP headers, then the WS-Security authentication types for all Web service operations must be set to None. You set the WS-Security authentication types in the Operations applets of the Inbound Web Services or Outbound Web Services views in the Administration-Web Services screen.

Session and Session Token Timeout-Related Parameters

You control the session timeout length and session token timeout length and maximum age by setting the parameters listed in the following table. These parameters are set in the eai application section of the AI profile.

Parameter Name	Parameter Value	Description
SessionTimeout	Number in seconds	The total number of seconds a session can remain inactive before the user is logged out and the session is closed. The default value is 900 seconds (15 minutes).

Parameter Name	Parameter Value	Description
GuestSessionTimeout	Number in seconds	The total number of seconds a guest session can remain inactive before the guest is logged out and the session is closed. The default value is 300 seconds (5 minutes).
SessionTokenTimeout	Number in minutes	The Siebel Application Interface (AI) rejects the session token if the token is inactive for more than the SessionTokenTimeout value. Whenever the token is used, this value is refreshed. You typically set SessionTokenTimeout to the same length of time as the global parameter SessionTimeout, whose default is 900 seconds (15 minutes). The default value is 15 minutes.
SessionTokenMaxAge	Number in minutes	The SessionTokenMaxAge parameter will make the AI reject the token if it has been used for more than the SessionTokenMaxAge value (for example, 240 minutes, or 4 hours). This is different from the SessionTokenTimeout because it does not refresh every time the token is used. The default value is 2880 minutes (two days).

Note: If you set the value of SessionTokenTimeout longer than the value of SessionTimeout and send a Web service request after the session times out, then a relogin occurs and the request is executed.

For information on SessionTimeout, see *Siebel Security Guide* . For information on application configuration parameters in general, see *Siebel System Administration Guide* .

Examples of Using SOAP Headers for Authentication and Session Management

The following examples illustrate using Siebel Authentication and Session Management SOAP headers. These examples use various authentication and session type combinations. For more information, see *Combinations of Session Types and Authentication Types*.

Anonymous Request No Session

This example illustrates an anonymous request and a session type of None, which closes the session after the response is sent out:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header>
    <SessionType xmlns="http://siebel.com/webservices">None</SessionType>
</soap:Header>
<soap:Body>
    <!-- data goes here -->
</soap:Body>
</soap:Envelope>
```

Siebel Authorization No Session

This example illustrates a request that includes authentication credentials (username and password) and a session type of None, which closes the session after the response is sent out:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header>
    <UsernameToken xmlns="http://siebel.com/webservices">user</UsernameToken>
    <PasswordText xmlns="http://siebel.com/webservices">hello123</PasswordText>
    <SessionType xmlns="http://siebel.com/webservices">None</SessionType>
</soap:Header>
<soap:Body>
    <!-- data goes here -->
</soap:Body>
</soap:Envelope>
```

Siebel Authorization Stateless Session

The following examples illustrate a request, response, and subsequent request for a session type set to Stateless, which keeps the session open after the initial response is sent out.

Initial Request

This example illustrates the initial request that includes authentication credentials (username and password) needed to log in:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header>
    <UsernameToken xmlns="http://siebel.com/webservices">user</UsernameToken>
    <PasswordText xmlns="http://siebel.com/webservices">hello123</PasswordText>
    <SessionType xmlns="http://siebel.com/webservices">Stateless</SessionType>
</soap:Header>
<soap:Body>
    <!-- data goes here -->
</soap:Body>
</soap:Envelope>
```

Response

This example illustrates the session token (encrypted) generated by the Siebel Server and sent back in the SOAP header of the response:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header>
    <siebel-header:SessionToken xmlns:siebel-header="http://siebel.com/
    webservices">2-r-JCunnMN9SxI9Any9zGQTOFIuJEJfCXjfI0G-

 9ZOOH4lJjbSd2P.G7vySzo07sFeJxUA0WhdnK_
    </siebel-header:SessionToken>
</soap:Header>
<soap:Body>
    <!-- data goes here -->
</soap:Body>
</soap:Envelope>
```

Subsequent Request Using Session Token

This example illustrates a subsequent request that includes the encrypted session token that was generated by the Siebel Server and passed in a previous response. The session token includes the user credentials and session information needed to reconnect to an existing session, or log in to a new one if the initial session has been closed:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header>
```

```
        <SessionType xmlns="http://siebel.com/webservices">Stateless</SessionType>
        <SessionToken xmlns="http://siebel.com/webservices">
        2-r-JCunnMN9SxI9Any9zGQTOFIuJEJfCXjfI0G-9ZOOH4lJjbSd2P.G7vySzo07sFeJxUA0WhdnK_
        </SessionToken>
    </soap:Header>
    <soap:Body>
        <!-- data goes here -->
    </soap:Body>
    </soap:Envelope>
```

Siebel Authorization Stateful Session

The following examples illustrate a request, response, and subsequent request for a session type set to Stateful, which keeps the session open after the initial response is sent out.

Initial Request

This example illustrates the initial request that includes authentication credentials (username and password) needed to log in:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header>
    <UsernameToken xmlns="http://siebel.com/webservices">user</UsernameToken>
    <PasswordText xmlns="http://siebel.com/webservices">hello123</PasswordText>
    <SessionType xmlns="http://siebel.com/webservices">Stateful</SessionType>
</soap:Header>
<soap:Body>
    <!-- data goes here -->
</soap:Body>
</soap:Envelope>
```

Response

This example illustrates the session token (encrypted) generated by the Siebel Server and sent back in the SOAP header of the response:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header>
    <siebel-header:SessionToken xmlns:siebel-header="http://siebel.com/
    webservices">Q7ABhvXBNUX5qTIoKJ9hZjhMzJ6lfTPa0oUDYxOBHkmOXB7j
    </siebel-header:SessionToken>
</soap:Header>
<soap:Body>
    <!-- data goes here -->
</soap:Body>
</soap:Envelope>
```

Subsequent Request Using Session Token

This example illustrates a subsequent request that includes the encrypted session token that was generated by the Siebel Server and passed in a previous response. The session token includes the user credentials and session information needed to reconnect to an existing session:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header>
    <SessionType xmlns="http://siebel.com/webservices">Stateful</SessionType>
    <SessionToken xmlns="http://siebel.com/
    webservices">Q7ABhvXBNUX5qTIoKJ9hZjhMzJ6lfTPa0oUDYxOBHkmOXB7j
    </SessionToken>
</soap:Header>
<soap:Body>
    <!-- data goes here -->
</soap:Body>
```

```
</soap:Envelope>
```

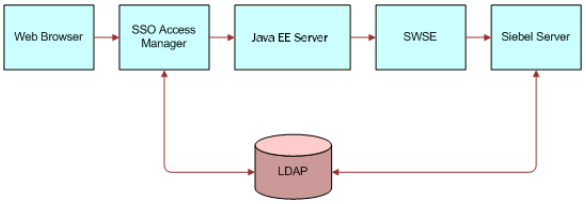
Simple Query Starting With <soap:body>

This example illustrates data for a simple query starting with the `<soap:body>` element:

```
<soap:body>
 <Account_spcService_Account_spcServiceQueryPage_Input
 xmlns="http://siebel.com/CustomUI">
 <ListOfTestAccount
 xmlns="http://www.siebel.com/xml/Test%20Account/Query">
 <Account>
 <Name>A*</Name>
 </Account>
 </ListOfTestAccount>
 </Account_spcService_Account_spcServiceQueryPage_Input>
</soap:body>
```

About Web Services and Web Single Sign-On Authentication

Siebel Web services support Web single sign-on (SSO) deployment scenarios in which third-party applications handle authentication, and then pass authentication information to the Siebel application. When the third-party application authenticates it, users do not have to explicitly log in to the Siebel application. The following illustrates a Web single SSO deployment scenario using Siebel Web services. For more information about Web SSO, see *Siebel Security Guide* .



Each component in the SSO scenario shown in the previous figure is described in the following:

- **SSO Access Manager.** SSO Access Manager, configured in front of the Java EE server, challenges user login, authenticates user credentials with LDAP, and sets a security token in the browser (http header), which is forwarded to the Java EE server.
- **Java EE Server.** This server extracts user credentials from the security token in the request. The Session Manager Login method takes the request as an argument and forwards it to the AI. The request contains the security token in the header.
- **AI.** AI extracts the user credentials from the security token and sends user credentials and the trust token to the Siebel Server.
- **Siebel Server.** The Siebel Server validates user credentials with LDAP and validates the trust token with security settings.

About SOAP Fault Schema Support

Service-Oriented Architecture (SOA) applications typically use Web services to expose functionality. The application describes a Web service through a WSDL document that is published. This WSDL document carries information about the input and output schema for each operation.

A client that invokes the Web service can use this WSDL document to determine the format of the request and response messages. Request and response messages are in SOAP format.

Siebel CRM consumes external Web services by processing the WSDL document and creating proxy business services. These proxy business services send requests to the external application and receive responses in a SOAP format. The responses are presented to the caller as Siebel property sets.

The WSDL document can optionally give a list of named faults (and their schema) that can occur for each operation. If an application error occurs, then the SOAP Fault element is used to capture it. The SOAP Fault element in the SOAP response body defines the following four subelements:

- **faultcode.** Identifies the fault.
- **faultstring.** Displays text that describes the fault.
- **faultactor.** Indicates the source of the fault.
- **detail.** Encodes application-specific errors.

The following WSDL example, which shows named faults, is from

http://www.gridlab.org:

```
 <?xml version="1.0" encoding="UTF-8"?>
 <definitions name="MyService" targetNamespace="urn:myuri:1.0"
xmlns:tns="urn:myuri:1.0"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="urn:myuri:1.0"
xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:MIME="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:DIME="http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/"
xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
 <schema targetNamespace="urn:myuri:1.0"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="urn:myuri:1.0"
xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified"
attributeFormDefault="unqualified">
<import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
<!-- fault element -->
<element name="MyFirstException">
 <complexType>
<sequence>
 <element name="text" type="xsd:string" minOccurs="1" maxOccurs="1"
 nillable="false"/>
</sequence>
```

```
 </complexType>
</element>
<!-- fault element -->
<element name="MySecondException">
 <complexType>
<sequence>
 <element name="number" type="xsd:int" minOccurs="1" maxOccurs="1"/>
</sequence>
 </complexType>
</element>
<!-- operation request element -->
<element name="myOperation">
 <complexType>
<sequence>
 <element name="myInput" type="xsd:string" minOccurs="0" maxOccurs="1"
 nillable="true"/>
</sequence>
 </complexType>
</element>
<!-- operation response element -->
<element name="myOperationResponse">
 <complexType>
<sequence>
 <element name="myOutput" type="xsd:string" minOccurs="0" maxOccurs="1"
 nillable="true"/>
</sequence>
 </complexType>
</element>
 </schema>
</types>
<message name="myOperationRequest">
<part name="parameters" element="ns1:myOperation"/>
</message>
<message name="myOperationResponse">
<part name="parameters" element="ns1:myOperationResponse"/>
</message>
<message name="MyFirstExceptionFault">
<part name="fault" element="ns1:MyFirstException"/>
</message>
<message name="MySecondExceptionFault">
<part name="fault" element="ns1:MySecondException"/>
</message>
<portType name="MyType">
 <operation name="myOperation">
<documentation>Service definition of function ns1__myOperation</documentation>
<input message="tns:myOperationRequest"/>
<output message="tns:myOperationResponse"/>
<fault name="MyFirstException" message="tns:MyFirstExceptionFault"/>
<fault name="MySecondException" message="tns:MySecondExceptionFault"/>
 </operation>
</portType>
<binding name="MyService" type="tns:MyType">
 <SOAP:binding style="document"
 transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="myOperation">
<SOAP:operation soapAction=""/>
<input>
<SOAP:body use="literal"/>
</input>
<output>
<SOAP:body use="literal"/>
</output>
<fault name="MyFirstException">
<SOAP:fault name="MyFirstException" use="literal"/>
</fault>
<fault name="MySecondException">
```

```
<SOAP:fault name="MySecondException" use="literal"/>
</fault>
 </operation>
</binding>
<service name="MyService">
 <documentation>gSOAP 2.7.1 generated service definition</documentation>
 <port name="MyService" binding="tns:MyService">
<SOAP:address location="http://localhost:10000"/>
 </port>
</service>
 </definitions>
```

The following SOAP message shows the first named fault from the example WSDL:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:ns1="urn:myuri:1.0">
 <SOAP-ENV:Body>
<SOAP-ENV:Fault>
 <faultcode>SOAP-ENV:Client</faultcode>
 <faultstring>Deliberately thrown exception.</faultstring>
 <detail>
<ns1:MyFirstException>
 <text>Input values are wrong.</text>
</ns1:MyFirstException>
 </detail>
</SOAP-ENV:Fault>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

For more information on fault handling in Web services, see the SOAP and WSDL standards listed in *Supported Web Services Standards*.

Handling SOAP Faults in Siebel CRM

The fault schemas described in the WSDL are accepted and modeled in Siebel CRM as integration objects. This is similar to how other input and output messages are modeled as strings (simple type) or integration objects (complex type). These named faults are available as output parameters.

Having named faults available as output parameters is useful in SOA environments. For example, Business Process Execution Language (BPEL) can only send named faults.

Handling SOAP Messages

SOAP messages are handled as follows:

1. An end user calls a proxy business service, for example, by clicking a button.
2. The proxy business service reads the input parameters and invokes the external Web service.
3. The proxy business service reads the SOAP response and checks for a SOAP Fault element.
4. If no SOAP fault is found, then the message is handled as normal.
5. If a SOAP fault is found, then the proxy business service tries to match the fault with a SOAP fault integration object.
    a. If a match is found, then Siebel converts the fault into a fault integration object instance and sets it in the output parameter.
    b. Whether or not a match is found, Siebel puts the fault into the XML Hierarchy (for backward compatibility).

Handling WSDL Imports

The handling of SOAP faults while importing WSDL files into Siebel CRM is as follows:

1. A developer uses the WSDL Import Wizard in Siebel Tools to import a WSDL document for creating proxy business services.
2. If the operation has a named fault that is not defined in the WSDL, then it is put into the XML Hierarchy.
3. If the operation has a named fault defined in the WSDL:
    a. If the Process Fault Schema check box is not selected, the named fault is ignored.
    b. If the Process Fault Schema check box is selected and an existing fault integration object is specified, then that fault integration object is added as an output parameter.
    c. If the Process Fault Schema check box is selected and an existing fault integration object is not specified, then a new fault integration object is added as an output parameter. The integration object name is prepended with Fault_.

    For information on using the WSDL Import Wizard, see *Creating an Outbound Web Service Based on a WSDL File*.

About Custom SOAP Filters

Headers represent SOAP's extensibility mechanism and provide a flexible and standards-based mechanism of adding additional context to a request or response. Custom SOAP header support provides a flexible extensibility mechanism when integrating with external Web services, and a means of providing additional context as required by the Web service implementation.

Handling Custom Headers Using Filters

SOAP headers provide the option of providing optional or mandatory processing information. To process optional custom headers that are provided by external applications, a special business service known as a filter might be defined. Filters can process both request and response headers. A special attribute, mustUnderstand, is used to indicate whether or not the custom header is to be processed:

- If 'mustUnderstand' equals 1, then the custom header is interpreted as being mandatory and the custom header is processed by the filter defined for this purpose.
- If 'mustUnderstand' equals 1 and a filter is not specified, then the custom header is not read and a SOAP:MustUnderstand fault is generated.
- If 'mustUnderstand' equals 0, then no processing of the custom header is attempted.

You must keep SOAP body and header processing isolated. The inbound dispatcher and outbound proxy can process the SOAP body, but cannot set or consume headers. Headers are application-specific. Some customization is needed to set and consume custom headers. To process optional custom headers that are provided by external applications, a special business service, a filter, is defined. You can configure the Web service outbound proxy and the Web service inbound dispatcher to call specific filters for the processing of individual (custom) headers.

> **Note:** The SOAP header will not be passed to the underlying business service or workflow of the inbound Web service. Any processing that must be done with the SOAP header must be done on the filter business service.

Enabling SOAP Header Processing Through Filters

For each operation, you can set the inbound and outbound filters to be run. You can also define the methods you want to call on the filter.

The following code sample illustrates a filter that has been written for the handling of custom SOAP headers. The interface provided by this code sample lets you define the method on the filter that you want to call, as well as the corresponding input and output parameters.

```
 function Service_PreInvokeMethod (MethodName, Inputs, Outputs) {
if(MethodName == "StripHeader") {
 if(Inputs.GetChildCount() > 0) {
 // Set the input SOAP message property set as the output.
 Outputs.InsertChildAt(Inputs.GetChild(0), 0);
 var soapEnv = Outputs.GetChild(0);
 if(soapEnv.GetChildCount() == 2) // headers and body {
// Here is where the header is found and processed.
var count = soapEnv.GetChildCount();
var i = 0;
var headerParam1;
var headerParam2;
var headerParam3;
// Use a loop just in case the header is not the first hierarchy.
for (; i < count; i++) {
 // For simplicity, the string comparison must be done using the exact same value

 // as the SOAP message tag name.
 if (soapEnv.GetChild(i).GetType() == "soapenv:Header") {
// Found the header. Now it is processed.
var soapHeader = soapEnv.GetChild(i);
// This example assumes that the following header hierarchy is received:
// <soapEnv:Header>
// <headerParam1>Value1</headerParam1>
// <headerParam2>Value2</headerParam2>
// <headerParam3>Value3</headerParam3>
// </soapEnv:Header>
// The parameters headerParam1, headerParam2, and headerParam3
// are saved into variables. Nothing further done with them.
headerParam1 = soapHeader.GetChild(0);
headerParam2 = soapHeader.GetChild(1);
headerParam3 = soapHeader.GetChild(2);
break; // Stop the loop after the header is found.
}
 }
 // Must remove the header from the SOAP property set.
 soapEnv.RemoveChild(i);
 }
}
 }
else if(MethodName == "AddHeader") {
 if(Inputs.GetChildCount() > 0) {
// Create the SOAP header hierarchy with the desired data.
var soapHeader = TheApplication().NewPropertySet();
soapHeader.SetType("soapEnv:Header");
soapHeader.SetProperty("xmlns:soapEnv",

"http://schemas.xmlsoap.org/soap/envelope/");
// These will be created as property sets because we want the following header:
// <soapEnv:Header>
// <headerParam1>Value1</headerParam1>
// <headerParam2>Value2</headerParam2>
```

```
// <headerParam3>Value3</headerParam3>
// </soapEnv:Header>
var param1PS = TheApplication().NewPropertySet();
var param2PS = TheApplication().NewPropertySet();
var param3PS = TheApplication().NewPropertySet();
param1PS.SetType("headerParam1");
param1PS.SetValue("Value1");
param2PS.SetType("headerParam2");
param2PS.SetValue("Value2");
param3PS.SetType("headerParam3");
param3PS.SetValue("Value3");
// Add the data to the SOAP header.
soapHeader.AddChild(param1PS);
soapHeader.AddChild(param2PS);
soapHeader.AddChild(param3PS);
// Get the SOAP envelope from the SOAP hierarchy.
var soapEnv = Inputs.GetChild(0);
// Add the header to the SOAP envelope.
soapEnv.InsertChildAt(soapHeader, 0);
Outputs.InsertChildAt(soapEnv, 0);
}
}
return(CancelOperation);
 }
```

Inputting a SOAP Envelope to a Filter Service

Using a SOAP envelope as the input to a filter service is the property set representation of an XML document. For example, each tag in the XML document is a property set. Each attribute on the tag is a property in the property set.

To pass the information in the headers further down the stack to the actual business service method or workflow being called, the *HeaderContext* property set is passed to the business service or workflow that is called. For example, on a call to an inbound Web service, if there are a couple of headers in the SOAP message, the filter service extracts the header information. To use this information in the business service or workflow execution call, it has to be contained in the *HeaderContext*. Internally, the Siebel Web services infrastructure passes *HeaderContext* to the eventual business service or workflow that is called.

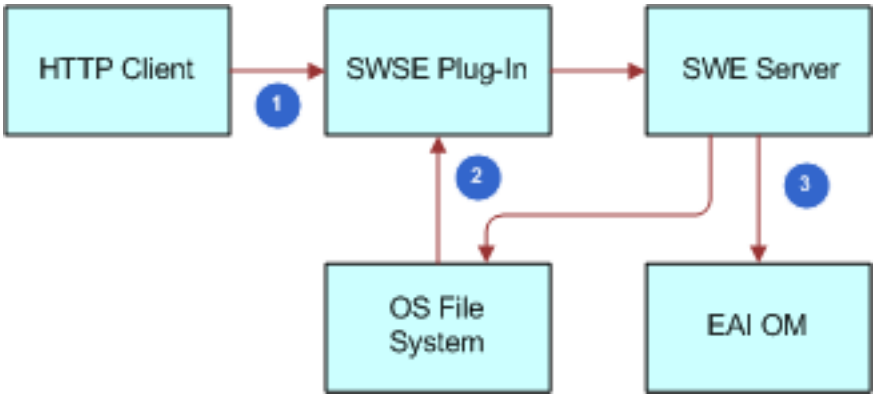
About EAI File Streaming

Siebel CRM supports streaming of EAI requests and responses encountered. This feature allows the Siebel Web Engine (SWE) and the EAI Object Manager (OM) to process Web service calls that involve large requests or responses. Large requests and responses can occur when inserting or querying file attachments by way of a Web service. By transferring data internally in 100-KB chunks, the memory footprints of the Siebel Web Engine and EAI Object Manager processes are reduced and system scalability is improved.

This topic describes the streaming process for inbound EAI requests and outbound responses.

About Inbound EAI Streaming Requests

The following figure provides an overview of the components and process flow used for streaming an inbound request.

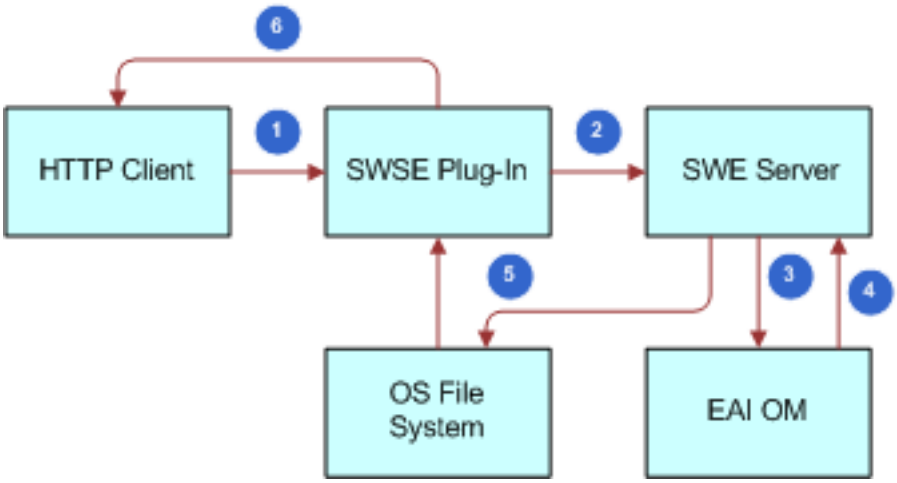


About Inbound EAI Streaming Requests illustrates the following process:

1. When the Application Interface (AI) Plug-in receives the request from the HTTP client by way of the Web Server, it determines that this request must be chunked based on the HTTP request body size, embeds streaming information in the request, and then sends the first chunk of the request body to the SWE Server.
2. The SWE Server extracts the streaming information from the request, determines it is a streaming request, and then writes the first chunk to the file system and sends a response to the plug-in indicating that the request has been processed. The AI sends the next chunk and the cycle continues until the last chunk has been written to the file system.
3. After the entire body of the HTTP request has been written to a file on the disk, the SWE Server calls the Web service method on EAI Object Manager, passing the name of the file as an input argument.

About Outbound EAI Streaming Responses

The following figure provides an overview of the components and process flow used for streaming an outbound response.



The previous figure illustrates the following process:

1. An EAI request from the HTTP client is received by the AI.
2. The AI forwards the request to the SWE Server.
3. The SWE Server then performs one of the following actions:

    a. If the request is not a streaming request, the SWE Server calls the EAI Object Manager method.

   b. If the request is a streaming request, then the file is first written to disk before the SWE Server calls the EAI Object Manager method.
4. The EAI Object Manager forwards the response to the SWE Server, and SWE queries the output arguments.
5. If a file reference is found, then the SWE Server transmits the file to the client.

   Whether multiple chunks will be sent or not depends upon the size of the file. If chunking is needed, then the SWE Server sends the first chunk to the plug-in, also embedding the streaming information in the response.
6. The Plug-in sends the chunk to the client including the HTTP headers in the response, and then it requeries the SWE Server to get the next chunk.

   Note: The plug-in sends the HTTP response headers only for the first chunk.

   The cycle continues until the entire file has been transmitted to the HTTP client.

About EAI Streaming Criteria

The following criteria are used internally to decide whether a particular request or response is streamed:

1. The Application Interface initiates chunking of an inbound EAI request only if the following conditions are met:

   a. The Configure EAI HTTP Inbound Transport parameter is selected in the basic information section of the AI profile of the eai application.
   b. The size of the body of the inbound HTTP request is greater than 100 KB.

   If these conditions are not met, then the plug-in does not stream the contents of the EAI request, and the request is processed as one chunk.
2. The SWE server initiates outbound chunking only if the following conditions are met:

   a. The SWE finds a property named ExtSvcFileName in the output arguments after calling the EAI Object Manager method.

      The value of this property must be a fully qualified path, and the name of the response file is written to disk by EAI.
   b. The file size is greater than 100 KB.

   If these conditions are not met, then the plug-in does not stream the contents of the EAI response, and the request is processed as one chunk.

About Web Services Cache Refresh

Both Siebel inbound and outbound Web services are typically cached into memory on the Siebel Server. At times, administrators must update the definitions of these services to provide more current or correct functionality. Administrators have the ability to directly refresh the memory cache in real time, without stopping and restarting the Siebel server.

The Web services cache is used to store all the global administration information that can be manipulated in the Inbound and Outbound Web Service administration views.

The Clear Cache feature is a button on the Administration - Web Services screen. This feature is available for inbound and outbound Web services. When the administrator decides when the Web service configuration must be refreshed, he or she clicks Clear Cache.

When Clear Cache is clicked, the integration object and Web services definitions in the run-time database are invalidated. Object definitions are reloaded when requested in the client.

Enabling Web Services Tracing

You can enable Web services tracing on the Siebel Server to write all inbound and outbound SOAP documents to a log file.

To enable Web services tracing

1. Navigate to the Administration - Server Configuration screen, Servers view.

   The view that appears displays three different list applets. The first applet lists the Siebel Servers for the enterprise. The middle applet has three tabs: Components, Parameters and Events. The last applet has two tabs: Events and Parameters.
2. In the first list applet, select the Siebel Server that you want to configure.
3. In the middle applet, click the Components tab.

   This list applet contains the components for the Siebel Server selected in the first applet.

   Choose the relevant application object manager.
4. In the last applet, click the Parameters tab.

   This list applet contains the parameters for the Component selected in the middle applet.
5. Set the Log Level to 4 for any or all of the following Event Types.

Event Type	Alias	Description	Comment
Web Service Performance	`WebSvcPerf`	Web Service Performance Event Type	Used for performance logging
Web Service Outbound Argument Tracing	`WebSvcOutboundArgTrc`	Web Service Outbound Run-time Argument Tracing	Used for logging arguments to the outbound dispatcher
Web Service Outbound	`WebSvcOutbound`	Web Service Outbound Run-time Event Type	Used for run-time logging of outbound Web services
Web Service Loading	`WebSvcLoad`	Web Service Configuration Loading Event Type	Used for logging of the loading of Web services

Event Type	Alias	Description	Comment
Web Service Inbound Argument Tracing	`WebSvcInboundArgTrc`	Web Service Inbound Run-time Argument Tracing	Used for logging arguments to the inbound dispatcher
Web Service Inbound	`WebSvcInbound`	Web Service Inbound Run-time Event Type	Used for logging at Web service inbound run time. Information is logged to the inbound dispatcher
Web Service Design	`WebSvcDesign`	Web Service Design-time Event Type	Used for logging at Web service design time. For example, at the time of WSDL import and generation

6. In the middle applet, click the Components tab.
7. Select the EAI Object Manager component, and then click the Parameters tab.

   The Component Parameters list appears.
8. Click Advanced to see the advanced parameters. (Click Reset to hide them again.)
9. Query for Enable Business Service Argument Tracing.
10. Set its Value and Value on Restart fields to True.
11. Restart or reconfigure the server component.

    For information on restarting server components and on advanced and hidden parameters, see *Siebel System Administration Guide* .

Previewing the Repository Changes Before Delivery

You can preview changes to a developer or integration workspace even before you deliver them. This is due to the `Workspace&Version` parameter, which helps preview changes specific to an object in a developer branch and/or its version. To learn more about using this parameter, see *Siebel REST API Guide* .

This topic gives an example of a SOAP request and response before and after changes to the Related Contact integration component (IC) in a developer workspace.

SOAP URI: `https://<host_name>:<port_number>/siebel/app/eai/enu?SWEExtSource=WebService&SWEExtCmd=Execute&WSSOAP=1`

SOAP Body:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:asi="http://siebel.com/asi/">
 <soapenv:Header>
 <ns1:SessionType xmlns:ns1="http://siebel.com/webservices">Nonens1:SessionType xmlns:ns1="http://siebel.com/webservices">None</ns1:SessionType>
 <ns2:UsernameToken xmlns:ns2="http://siebel.com/webservices"><username>:UsernameToken xmlns:ns2="http://siebel.com/webservices"><username></ns2:UsernameToken>
```

```
 <ns3:PasswordText xmlns:ns3="http://siebel.com/webservices"><password>:PasswordText xmlns:ns3="http://
siebel.com/webservices"><password></ns3:PasswordText>
 </soapenv:Header>
 <soapenv:Body>
 <asi:SiebelAccountQueryById_Input>
 <asi:PrimaryRowId><Primary Row Id></asi:PrimaryRowId>
 </asi:SiebelAccountQueryById_Input>
 </soapenv:Body>
</soapenv:Envelope>
```

When you request details, the Related Contacts IC displays along with other objects in the workspace:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <SOAP-ENV:Body>
 <ns:SiebelAccountQueryById_Output xmlns:ns="http://siebel.com/asi/">
 <ListOfAccountInterface xmlns="http://www.siebel.com/xml/Account%20Interface">
 <Account>
 .
 .
 .
 <ListOfRelatedSalesRep>
 .
 .
 .
 </ListOfRelatedSalesRep>
 <ListOfRelatedContact>
 <RelatedContact>
 <ContactId><Contact IdContact Id>ContactId>
 <FirstName>JohnFirstName>John>
 <ContactIntegrationId/>
 <LastName>Smith FINSLastName>Smith FINS>
 <MiddleName/>
 <PersonUId>Contact IdPersonUId>Contact Id>
 <PrimaryOrganization>ABC Insurance IN ENUPrimaryOrganization>ABC Insurance IN ENU>
 </RelatedContact>
 <RelatedContact>
 .
 .
 .
 </RelatedContact>
 </ListOfRelatedContact>
 <ListOfRelatedOrganization>
 .
 .
 .
 </ListOfRelatedOrganization>
 .
 .

 </Account>
 </ListOfAccountInterface>
 </ns:SiebelAccountQueryById_Output>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

You can modify the object definition and then preview the changes before delivery of workspace. In the following example, Related Contact is no longer a child component of the base List of Related Contact integration object (IO). This change to the structure of the IO is seen by passing the workspace name and its version as query parameters in the request. The response fetches other objects but not the Related Contacts IC and its IO due to inactivation of the IC.

For more information about how to inactivate an object in Web Tools, see *Using Siebel Tools* . For more information on how to activate a web service, see *Invoking Siebel Web Services Using an External System*.

SOAP URI: `https://<host_name>:<port_number>/siebel/app/eai/enu?SWEExtSource=WebService&SWEExtCmd=Execute&WSSOAP=1&Workspace=<Workspace_<workspace_name>&Version=<ver_num>>`

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 </soapenv:Body>
 <ns:SiebelAccountQueryById_Output xmlns:ns="http://siebel.com/asi/">
 <ListOfAccountInterface xmlns="http://www.siebel.com/xml/Account%20Interface">
 <Account>
 .
 .
 .
 <ListOfRelatedSalesRep>
 .
 .
 .
 </ListOfRelatedSalesRep>
 <ListOfRelatedOrganization>
 .
 .
 .
 </ListOfRelatedOrganization>
 </Account>
 </ListOfAccountInterface>
 </ns:SiebelAccountQueryById_Output>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

6 EAI Siebel Adapter Business Service

EAI Siebel Adapter Business Service

EAI Siebel Adapter is a preconfigured business service that is used with any integration process that runs through the Siebel business object layer. Integration objects are used to update data in business objects and are used when retrieving data from business objects. These integration objects are configurable and can be used during an integration process (for example, entering and retrieving data from Siebel CRM).

This chapter describes the functionality of the EAI Siebel Adapter business service, and the different methods and arguments you can use with it to manipulate the data in the Siebel Database. It includes:

- *About the EAI Siebel Adapter Business Service*
- *EAI Siebel Adapter Business Service Methods*
- *Skipnode Operation*
- *About the Search Spec Input Method Argument*
- *About Using Effective Dating with Siebel EAI Adapter Business Service*
- *Enabling Effective Dating on Fields*
- *Enabling Effective Dating on Links*
- *About Using Language-Independent Code with the EAI Siebel Adapter Business Service*
- *Siebel EAI and Run-Time Events*
- *Guidelines for Using the EAI Siebel Adapter Business Service*
- *Troubleshooting the EAI Siebel Adapter Business Service*
- *Enabling Logging for the EAI Siebel Adapter Business Service*
- *Enabling Siebel Argument Tracing*
- *Configuring the EAI Siebel Adapter Business Service for Concurrency Control*

About the EAI Siebel Adapter Business Service

EAI Siebel Adapter is a general-purpose integration business service that allows you to:

- Read Siebel business objects from the Siebel Database into integration objects.

  > **Note:** When called locally, the EAI Siebel Adapter business service creates an additional database connection. If this second connection times out, then further transactions will not be processed. To prevent this from happening, add the TrxDbConnReconnectIntervalSeconds parameter to the [ServerDataSrc] section of the application configuration (CFG) file. A typical value is 120.

- Write an integration object instance whose data originates externally in a Siebel business object.
- Update multiple corresponding top-level (highest-level) parent business component records with data from one XML file, for an example, see *Update Method*.

> **Note:** The Siebel Message is considered to be one transaction. The transaction is committed when there is no error. If there is an error, then the transaction is aborted and rolled back.

Node Types and the EAI Siebel Adapter Business Service

In an integration object hierarchy, nodes with at least one child are called internal nodes and nodes without children are called leaf nodes. When either the insert or update method is called on the EAI Siebel Adapter business service, the adapter performs the operation on both internal nodes and leaf nodes. When the insert or update method is called on the EAI UI Data Adapter business service, the adapter performs insert on leaf nodes only.

For more information on node types, see *About the EAI UI Data Adapter Business Service*.

EAI Siebel Adapter Business Service Methods

The EAI Siebel Adapter supports the following methods:

- *Query Method*
- *QueryPage Method*
- *Synchronize Method*
- *Insert Method*
- *Upsert Method*
- *Update Method*
- *Delete Method*
- *Execute Method*

About the Examples in the EAI Siebel Adapter Business Service Methods Sections

The following information is true for the examples used for the EAI Siebel Adapter methods:

- The business object data is represented as integration object data in XML format.
- The XML document or integration object instance might also be referred to as a Siebel Message.
- Fields that contain null values are not included in the XML examples.
  However, these fields might be revealed when you use EAI XML Write to File.WriteEAIMsg() to print out the XML.

Query Method

You can use a combination of input arguments when using the Query Business Service Method of the EAI Siebel Adapter. The input arguments are as follows:

1. **Query By Example (QBE).** Pass in an integration object instance represented as a property set.

> **Note:** EAI queries on integration objects do not use a search specification for child integration components when the query obtains the parent integration component.

2. **Primary Row Id.** Pass in a string to the Object Id input argument. The string can be the row_id of the primary business component of the Output Integration Object Name.
3. **Output Integration Object Name.** See the Primary Row Id for information.
4. **Search Specification.** Pass in a String expression.

The input arguments can be used in one of the following combinations:

- 1
- 2 and 3
- 4
- 3 and 4
- 2, 3, and 4

The EAI Siebel Adapter uses this input as criteria to query the base business object and to return a corresponding integration object instance.

For an example of using the search specification method argument to limit the scope of your query see *About Using Language-Independent Code with the EAI Siebel Adapter Business Service*.

When using the EAI Siebel Adapter, to query all the business component records, you are not required to specify any value in the Object Id process property of the workflow. In this case, not specifying an Object Id or a Search Specification works as a wildcard.

If you want to query Siebel data using the EAI Siebel Adapter with the Query method and an integration object instance (property set) containing a query by example (QBE) search criterion, then all the fields present in the QBE will be used in the query. To retrieve a unique record, include the fields that make up the user key for the underlying integration object component instance to ensure you retrieve a unique record. You can use an asterisk (*) as a wildcard for each one of the fields.

For example, the following is your QBE:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2IOY" IntObjectName = "EAI Account" MessageType =
"Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfAccount>
 <Account>
 <CSN>*</CSN>
 <HomePage>*</HomePage>
 <Location>H*</Location>
 <Name>A*</Name>
 <Type>*</Type>
 </Account>
 </ListOfAccount>
</SiebelMessage>
```

You would receive all of the Accounts with names that start with A* and have locations that start with H*. The CSN, HomePage, and Type fields cannot be blank because they are used in the query.

The EAI Siebel Adapter converts the QBE into a user Search Expression of the following:

```
[CSN] ~ LIKE "*" AND [Home Page] ~ LIKE "*" AND [Location] ~ LIKE "H*" AND [Name] ~
LIKE "A*" AND [Type] ~ LIKE "*"
```

You can run this example and review the output XML generated.

When using search expressions that contain an apostrophe ('), you must use two apostrophes (") or the search will fail.

For example, you are searching for the string LUKE'S. The following search expression will fail in the EAI Siebel Adapter:

```
[Account.Name] LIKE "*LUKE'S*"
```

Use the following search expression instead:

```
[Account.Name] LIKE "*LUKE''S*"
```

Note: The EAI Siebel Adapter explicitly overrides any Object Manager settings for the MaxCursorSize parameter. The EAI Siebel Adapter uses a MaxCursorSize of -1. If you want to limit the number of results received when using the Query method, then use the QueryPage Method. You can combine the Object Id and Search Specification together to query for parent and child data.

Note: The EAI Siebel Adapter returns the output of the Query() method as one Siebel Message. This integration object instance is stored in the process memory. If your query returns a large number of records, this will result in your Siebel component's memory consumption being high.

QueryPage Method

This method is useful when the search specification retrieves a large number of records at the root component. To avoid returning one huge Siebel Message, you can specify the number of records to be returned using the PageSize argument, as presented in *Skipnode Operation*. You can also use method arguments such as OutputIntObjectName, SearchSpec, SortSpec, ViewMode, and StartRowNum to dictate which records to return.

Even though the QueryPage returns a limited number of records, it keeps the data in the cache, which you can then retrieve by calling the EAI Siebel Adapter with a new value for the StartRowNum method argument. Note that this is only possible if the method arguments OutputIntObjectName, PageSize, SearchSpec, SortSpec, and ViewMode are not changed and the NewQuery method argument is set to False.

Note: The EAI Siebel Adapter returns the output of the QueryPage() method as one Siebel Message. This integration object instance is stored in the process memory. If your query returns a large number of records, this will result in your Siebel component's memory consumption being high.

The QueryPage method precedes each integration object instance. It is provided through the SiebelMessage input argument when performing a query by example. Parameters such as StartRowNum, PageSize, and others are applied to each integration object instance.

For example, a Siebel database contains four account records with the Name field set to: a1, a2, b1, b2. An input SiebelMessage with two instances of the Account integration object with the first instance's name set to "a*" and the second instance's name set to "b*" the result for StartRowNum=0 is all four records (a1, a2, b2, b4), and for StartRowNum=1 only two records (a2 and b2). This example illustrates that the StartRowNum method argument counts records within each single integration object instance of the query by example input SiebelMessage, once for "a*" (a1, a2) and once for "b*" (b1, b2).

The following is an example of using the QueryPage() method in a business service.

```
var EAIService = TheApplication().GetService("EAI Siebel Adapter");
var writeSvc = TheApplication().GetService("EAI XML Write to File");
var EAIin = TheApplication().NewPropertySet();
```

```
var ResultSet= TheApplication().NewPropertySet();
var moreRecords = true;
var countOfObjects = 0;
var i = 1;

// set up input arguments, get 10 at a time
EAIin.SetProperty("OutputIntObjectName", "EAI Account");
EAIin.SetProperty("PageSize", "10");
EAIin.SetProperty("SearchSpec", "[Account.Name] LIKE '3*'");
EAIin.SetProperty("StartRowNum", i);
EAIin.SetProperty("NewQuery", "true");

// retrieve the business component data
EAIService.InvokeMethod("QueryPage", EAIin, ResultSet);

// loop through cached data
while ( (ResultSet.GetChildCount() > 0) && (moreRecords)) {
countOfObjects = countOfObjects + ResultSet.GetProperty("NumOutputObjects");

// write out first chunk of data retrieved
ResultSet.SetProperty("FileName", "d:\\temp\\EAIaccount$$.xml");
writeSvc.InvokeMethod("WriteEAIMsg", ResultSet, Outputs);

// reuse the existing input property set, except don't reissue query
EAIin.SetProperty("NewQuery", "false");
i= i+10; // get next 10 records
EAIin.SetProperty("StartRowNum", i);

ResultSet.Reset(); // clear previous result set
EAIService.InvokeMethod("QueryPage", EAIin, ResultSet);
if (ResultSet.GetProperty("LastPage") == "true")
moreRecords = false;
}
```

Synchronize Method

You can use the Synchronize method to make the values in a business object instance match those of an integration object instance. This operation can result in updates, insertions, or deletions in the business components. The following rules apply to the results of this method:

- If a child component is not present in the integration object instance, then the corresponding child business component rows are left untouched.
- If the integration object instance's child component has an empty container, then all child records in the corresponding business component are deleted.
- For a particular child component, records that exist in both the integration object instance and business component are updated. Records that exist in the integration object hierarchy and not in the business component are inserted. Records in the business component and not in the integration object instance are deleted.
- Only the fields specified in the integration component instance are updated.

Note: When the EAI Siebel Adapter starts a database transaction (initiated to allow updates to the Siebel database) it must ensure the data queried is committed and consistent. The results of these queries dictate what changes are applied, so if reads that contain uncommitted data (*dirty reads*) were enabled, it could cause incorrect updates by the EAI Siebel Adapter. Therefore, dirty reads are disabled during database transactions started by the EAI Siebel Adapter.

Example of Synchronize Method on Deleted Unmatched Children

This first example demonstrates deleting unmatched children when using the Synchronize method. This example uses data present in the sample database.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType =
"Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfAccount>
<Account>
<AccountStatus>Active</AccountStatus>
<CSN>1-3JGO7</CSN>
<CurrencyCode>USD</CurrencyCode>
<LanguageCode>ENU</LanguageCode>
<Location>Test</Location>
<Name>ABC Corp</Name>
<ListOfAccount_BusinessAddress>
 <Account_BusinessAddress IsPrimaryMVG = "Y">
 <AddressActiveStatus>Y</AddressActiveStatus>
 <BillAddressFlag>Y</BillAddressFlag>
 <City>ATown</City>
 <Country>USA</Country>
 <MainAddressFlag>Y</MainAddressFlag>
 <ShipAddressFlag>Y</ShipAddressFlag>
 <StreetAddress>123 Main St</StreetAddress>
 </Account_BusinessAddress>
 <Account_BusinessAddress IsPrimaryMVG = "N">
 <AddressActiveStatus>Y</AddressActiveStatus>
 <BillAddressFlag>Y</BillAddressFlag>
 <City>BTown</City>
 <Country>USA</Country>
 <MainAddressFlag>Y</MainAddressFlag>
 <ShipAddressFlag>Y</ShipAddressFlag>
 <StreetAddress>456 Oak St</StreetAddress>
 </Account_BusinessAddress>
</ListOfAccount_BusinessAddress>
<ListOfContact>
 <Contact>
 <ActiveStatus>Y</ActiveStatus>
 <FirstName>User1</FirstName>
 <LastName>User1</LastName>
 <Organization>Default Organization</Organization>
 <ListOfContact_Organization>
 <Contact_Organization IsPrimaryMVG = "Y">
 <Organization>Default Organization</Organization>
 <OrganizationIntegrationId/>
 </Contact_Organization>
 </ListOfContact_Organization>
 <ListOfContact_AlternatePhone/>
 </Contact>
 <Contact>
 <ActiveStatus>Y</ActiveStatus>
 <FirstName>User2</FirstName>
 <LastName>User2</LastName>
 <Organization>Default Organization</Organization>
 <ListOfContact_Organization>
 <Contact_Organization IsPrimaryMVG = "Y">
 <Organization>Default Organization</Organization>
 <OrganizationIntegrationId/>
 </Contact_Organization>
 </ListOfContact_Organization>
 <ListOfContact_AlternatePhone/>
 </Contact>
 </ListOfContact>
```

```
 <ListOfAccount_Organization>
 <Account_Organization IsPrimaryMVG = "Y">
 <Organization>Default Organization</Organization>
 <OrganizationId>0-R9NH</OrganizationId>
 <OrganizationIntegrationId/>
 </Account_Organization>
</ListOfAccount_Organization>
</Account>
 </ListOfAccount>
</SiebelMessage>
```

Then the following XML (integration object instance) is submitted with Synchronize:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType =
"Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfAccount>
<Account>
<AccountStatus>Active</AccountStatus>
<CSN>1-3JGO7</CSN>
<Competitor>Y</Competitor>
<CurrencyCode>USD</CurrencyCode>
<LanguageCode>CHS</LanguageCode>
<Location>test</Location>
<Name>ABC Corp</Name>
<ListOfContact>
 <Contact>
 <ActiveStatus>N</ActiveStatus>
 <FirstName>User1</FirstName>
 <LastName>User1</LastName>
 <MiddleName></MiddleName>
 <Organization>Default Organization</Organization>
 </Contact>
 <Contact>
 <FirstName>User3</FirstName>
 <LastName>User3</LastName>
 <MiddleName></MiddleName>
 <Organization>Default Organization</Organization>
 </Contact>
</ListOfContact>
</Account>
 </ListOfAccount>
</SiebelMessage>
```

Following is the result you will receive. Because the contact information is included in the integration object instance, User2 in the database is deleted because it was an unmatched node. User1 is updated because it is a matched node. User3 is inserted because it is a new node. Since Business Address was not included in the integration object instance, it is left in the business object.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType =
"Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfAccount>
<Account>
<AccountStatus>Active</AccountStatus>
<CSN>1-3JGO7</CSN>
<CurrencyCode>USD</CurrencyCode>
<LanguageCode>CHS</LanguageCode>
<Location>Test</Location>
<Name>ABC Corp</Name>
<ListOfAccount_BusinessAddress>
 <Account_BusinessAddress IsPrimaryMVG = "Y">
 <AddressActiveStatus>Y</AddressActiveStatus>
```

```
<BillAddressFlag>Y</BillAddressFlag>
<City>ATown</City>
<Country>USA</Country>
<MainAddressFlag>Y</MainAddressFlag>
<ShipAddressFlag>Y</ShipAddressFlag>
<StreetAddress>123 Main St</StreetAddress>
</Account_BusinessAddress>
<Account_BusinessAddress IsPrimaryMVG = "N">
<AddressActiveStatus>Y</AddressActiveStatus>
<BillAddressFlag>Y</BillAddressFlag>
<City>BTown</City>
<Country>USA</Country>
<MainAddressFlag>Y</MainAddressFlag>
<ShipAddressFlag>Y</ShipAddressFlag>
<StreetAddress>456 Oak St</StreetAddress>
</Account_BusinessAddress>
</ListOfAccount_BusinessAddress>
<ListOfContact>
<Contact>
<ActiveStatus>N</ActiveStatus>

 <FirstName>User1</FirstName>

 <LastName>User1</LastName>
<Organization>Default Organization</Organization>
<ListOfContact_Organization>
<Contact_Organization IsPrimaryMVG = "Y">
<Organization>Default Organization</Organization>
<OrganizationIntegrationId/>
</Contact_Organization>
</ListOfContact_Organization>
<ListOfContact_AlternatePhone/>
</Contact>
<Contact>
<ActiveStatus>N</ActiveStatus>
<FirstName>User3</FirstName>
<LastName>User3</LastName>
<Organization>Default Organization</Organization>
<ListOfContact_Organization>
<Contact_Organization IsPrimaryMVG = "Y">
<Organization>Default Organization</Organization>
<OrganizationIntegrationId/>
</Contact_Organization>
</ListOfContact_Organization>
<ListOfContact_AlternatePhone/>
</Contact>
</ListOfContact>
</Account>
 </ListOfAccount>
</SiebelMessage>
```

The following table is a high level representation of the previous example.

Record In Database	Integration Object Instance	Record After Synchronize
Account: ABC Corp • Business Address: 123 Main St • Business Address: 456 Oak St • Contact: User1	Account: ABC Corp • Contact: User1 • Contact: User3	Account: ABC Corp • Business Address: 123 Main St • Business Address: 456 Oak St • Contact: User1

Record In Database	Integration Object Instance	Record After Synchronize
○ Organization: Default Org. • Contact: User2 ○ Organization: Default Org. • Organization: Default Org.		• Organization: Default Org • Contact: User3 • Organization: Default Org. • Organization: Default Org.

This second example demonstrates how all records with an empty container are deleted when using the Synchronize method.

If you start with this business component data:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType =
"Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfAccount>
<Account>
<AccountStatus>Active</AccountStatus>
<CSN>1-3JGO7</CSN>
<CurrencyCode>USD</CurrencyCode>
<LanguageCode>ENU</LanguageCode>
<Location>test</Location>
<Name>ABC Corp</Name>
<ListOfAccount_BusinessAddress>
 <Account_BusinessAddress IsPrimaryMVG = "Y">
 <AddressId>1-3JGOA</AddressId>
 <AddressActiveStatus>Y</AddressActiveStatus>
 <BillAddressFlag>Y</BillAddressFlag>
 <City>MyTown</City>
 <Country>Canada</Country>
 <MainAddressFlag>Y</MainAddressFlag>
 <ShipAddressFlag>Y</ShipAddressFlag>
 <StreetAddress>123 Main St</StreetAddress>
 </Account_BusinessAddress>
 <Account_BusinessAddress IsPrimaryMVG = "N">
 <AddressActiveStatus>Y</AddressActiveStatus>
 <BillAddressFlag>Y</BillAddressFlag>
 <AddressId>1-3JGOB</AddressId>
 <City>YourTown</City>
 <Country>Canada</Country>
 <MainAddressFlag>Y</MainAddressFlag>
 <ShipAddressFlag>Y</ShipAddressFlag>
 <StreetAddress>456 Oak St</StreetAddress>
 </Account_BusinessAddress>
 </ListOfAccount_BusinessAddress>
 <ListOfContact>
 <Contact>
 <ActiveStatus>Y</ActiveStatus>
 <FirstName>User1</FirstName>
 <LastName>User1</LastName>
 <MiddleName/>
 <Organization>Default Organization</Organization>
 <ListOfContact_Organization>
 <Contact_Organization IsPrimaryMVG = "Y">
 <Organization>Default Organization</Organization>
 <OrganizationIntegrationId/>
 </Contact_Organization>
```

```
    </ListOfContact_Organization>
    <ListOfContact_AlternatePhone/>
    </Contact>
    <Contact>
    <ActiveStatus>Y</ActiveStatus>
    <FirstName>User2</FirstName>
    <LastName>User2</LastName>
    <MiddleName/>
    <Organization>Default Organization</Organization>
    <ListOfContact_Organization>
    <Contact_Organization IsPrimaryMVG = "Y">
    <Organization>Default Organization</Organization>
    <OrganizationIntegrationId/>
    </Contact_Organization>
    </ListOfContact_Organization>
    <ListOfContact_AlternatePhone/>
    </Contact>
    </ListOfContact>
    <ListOfAccount_Organization>
    <Account_Organization IsPrimaryMVG = "Y">
    <Organization>Default Organization</Organization>
    <OrganizationId>0-R9NH</OrganizationId>
    <OrganizationIntegrationId/>
    </Account_Organization>
  </ListOfAccount_Organization>
  </Account>
    </ListOfAccount>
  </SiebelMessage>
```

And the following integration object instance is passed in:

```
  <?xml version = "1.0" encoding = "UTF-8"?>
  <?Siebel-Property-Set EscapeNames="false"?>
  <SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType =
  "Integration Object" IntObjectFormat = "Siebel Hierarchical">
    <ListOfAccount>
    <Account>
    <AccountStatus>Active</AccountStatus>
    <CSN>1-3JGO7</CSN>
    <Competitor>Y</Competitor>
    <CurrencyCode>USD</CurrencyCode>
    <LanguageCode>CHS</LanguageCode>
    <Location>test</Location>
    <Name>ABC Corp</Name>
  <ListOfContact/>
  </Account>
    </ListOfAccount>
  </SiebelMessage>
```

After the sync operation, all the children contacts are deleted because none of the nodes match.

```
  <?xml version = "1.0" encoding = "UTF-8"?>
  <?Siebel-Property-Set EscapeNames="false"?>
  <SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType =
  "Integration Object" IntObjectFormat = "Siebel Hierarchical">
    <ListOfAccount>
  <Account>
    <AccountStatus>Active</AccountStatus>
    <CSN>1-3JGO7</CSN>
    <CurrencyCode>USD</CurrencyCode>
    <LanguageCode>ENU</LanguageCode>
    <Location>test</Location>
    <Name>ABC Corp</Name>
    <ListOfAccount_BusinessAddress>
    <Account_BusinessAddress IsPrimaryMVG = "Y">
    <AddressId>1-3JGOA</AddressId>
```

```
    <AddressActiveStatus>Y</AddressActiveStatus>
    <BillAddressFlag>Y</BillAddressFlag>
    <City>MyTown</City>
    <Country>Canada</Country>
    <MainAddressFlag>Y</MainAddressFlag>
    <ShipAddressFlag>Y</ShipAddressFlag>
    <StreetAddress>123 Main St</StreetAddress>
    </Account_BusinessAddress>
    <Account_BusinessAddress IsPrimaryMVG = "N">
    <AddressActiveStatus>Y</AddressActiveStatus>
    <BillAddressFlag>Y</BillAddressFlag>
    <AddressId>1-3JGOB</AddressId>
    <City>YourTown</City>
    <Country>Canada</Country>
    <MainAddressFlag>Y</MainAddressFlag>
    <ShipAddressFlag>Y</ShipAddressFlag>
    <StreetAddress>456 Oak St</StreetAddress>
    </Account_BusinessAddress>
    </ListOfAccount_BusinessAddress>
    <ListOfAccount_Organization>
    <Account_Organization IsPrimaryMVG = "Y">
    <Organization>Default Organization</Organization>
    <OrganizationId>0-R9NH</OrganizationId>
    <OrganizationIntegrationId/>
    </Account_Organization>
    </ListOfAccount_Organization>
  </Account>
   </ListOfAccount>
  </SiebelMessage>
```

The following table is a high level representation of the operation.

This second example demonstrates how all records with an empty container are deleted when using the Synchronize method.

Record In Database	Integration Object Instance	Record After Synchronize Operation
Account: ABC Corp • Business Address: 123 Main St • Business Address: 456 Oak St • Contact: User1 o Organization: Default Org. • Contact: User2 o Organization: Default Org. • Organization: Default Org.	Account: ABC Corp • Contact:	Account: ABC Corp • Business Address: 123 Main St • Business Address: 456 Oak St • Organization: Default Org.

Insert Method

This method is also similar to the Synchronize method with the exception that the EAI Siebel Adapter generates an error if a matching root component is found; otherwise, it inserts the root component and synchronizes all the children. It is

important to note that when you insert a record, there is a possibility that the business component would create default children for the record, which will be removed by the Insert method. The Insert method synchronizes the children, which deletes all the default children. For example, if you insert an account associated with a specific organization, then it will also be automatically associated with a default organization. As part of the Insert method, the EAI Siebel Adapter deletes the default association, and associates the new account with only the organization that was originally defined in the input integration object instance. The EAI Siebel Adapter achieves this by synchronizing the children.

Example of Using the Insert Method

If you use the Insert method with the example of the integration object instance represented in XML that follows, then a new service request is created with two activities.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2R6E" IntObjectName = "Sample Service Request"
MessageType = "Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfSampleServiceRequest>
<ServiceRequest>
 <Account>Genesys Communications</Account>
 <AccountLocation>San Francisco, CA</AccountLocation>
 <Area>Network</Area>
 <ClosedDate/>
 <CommitTime/>
 <ContactBusinessPhone>4155551100</ContactBusinessPhone>
 <ContactLastName>Kastrup-Larsen</ContactLastName>
 <Description>Setting up Router services<Description>
 <Priority>3-Medium</Priority>
 <SRNumber>1-MYUNIQUEVALUE</SRNumber>
 <ServiceRequestType>External</ServiceRequestType>
 <ListOfAction>
 <Action>
 <BillableFlag>N</BillableFlag>
 <Description2>test activity1</Description2>
 <EstWorkTimeRemaining>8</EstWorkTimeRemaining>
 <Planned/>
 <PrimaryOwnedBy>SADMIN</PrimaryOwnedBy>
 <RowStatusOld>N</RowStatusOld>
 <Status>Unscheduled</Status>
 <Type>Appointment</Type>
 </Action>
 <Action>
 <BillableFlag>N</BillableFlag>
 <Description2>test activity2</Description2>
 <EstWorkTimeRemaining>8</EstWorkTimeRemaining>
 <Planned/>
 <PrimaryOwnedBy>SADMIN</PrimaryOwnedBy>
 <Status>Unscheduled</Status>
 <Type>Appointment</Type>
 </Action>
 </ListOfAction>
 </ServiceRequest>
 </ListOfSampleServiceRequest>
</SiebelMessage>
```

For this example to work, you must have the contact, Kastrup-Larsen, in the database. If you try the Insert method against a server database where the contact does not exist, then you will receive the following error:

```
Picklist validation of field 'Contact Last Name' in integration component 'Service
Request' did not find any matches satisfying the query '[Last Name] = "Kastrup-
Larsen"', and an attempt to create a new record through the picklist failed (SBL-
EAI-04186)
```

Also, if you try to insert the previous instance a second time, then you will receive the following error message:

```
IDS_ERR_EAI_SA_INSERT_MATCH_FOUND. Insert operation on integration component
'Service Request' failed because a matching record in business component 'Service
Request' with search specification '[SR Number] = "1-MYUNIQUEVALUE" was found.(SBL-
EAI-04383).
```

Upsert Method

The Upsert method is similar to the Synchronize method with one exception; the Upsert method does not delete any records.

The Upsert method will result in insert or update operations. If the record exists, then it will be updated. If the record does not exist, then it will be inserted. Unlike the Synchronize method, upsert will not delete any children.

To determine if an update or insert is performed, the EAI Siebel Adapter runs a query using user keys fields or the search specifications to determine if the parent or primary record already exists. If the parent record exists, it will be updated. If no matching parent record is found, then the new record will be inserted. Once again, upsert will not delete any children. If existing children are found, then they are updated.

You can update multiple corresponding top-level (highest-level) parent business component records using one XML file, as in the following example:

```
<SiebelMessage MessageId="" MessageType="Integration Object"
 IntObjectName="Transaction">
 <ListofTransaction>
 <Transaction>
 <Field1>xxxx</Field1>
 <Field2>yyyy</Field2>
 ...
 </Transaction>
 <Transaction>
 <Field1>aaaa</Field1>
 <Field2>bbbb</Field2>
 ...
 </Transaction>
 ...
 </ListofTransaction>
 </SiebelMessage>
```

Update Method

This method is similar to the Synchronize method, except that the EAI Siebel Adapter returns an error if no match is found for the root component; otherwise, it updates the matching record and synchronizes all the children.

> **Note:** During an update operation, the EAI Siebel Adapter expects a single record to be returned from the user key search. If more than one record is returned, then EAI Siebel Adapter throws an error.

For example, if you send an order with one order item to the EAI Siebel Adapter, then it will take the following actions:

1. Queries for the order, and if it finds a match, then it updates the record.
2. Updates or inserts the new order item depending on whether a match was found for the new order item.
3. Deletes any other order items associated with that order.

Delete Method

You can delete one or more records in a business component that is mapped to the root integration component, given an integration object instance. A business component record is deleted as specified by an integration object instance. The integration component instance fields are used to query the corresponding business component and any records retrieved will be deleted. You call the Delete method using only one of the following method arguments:

- A Query By Example (QBE) integration object instance.
- A Primary Row Id and Output Integration Object Name.
- A Search Specification.

Note: To have the EAI Siebel Adapter perform a delete operation, define an integration object that contains the minimum fields on the primary business component for the business object. The EAI Siebel Adapter attempts to delete matching records in the business component before deleting the parent record.

For example, if you pass in this XML document, then the Test Account account is deleted.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2IOY" IntObjectName = "EAI Account" MessageType =
"Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfAccount>
 <Account>
 <Name>Test Account</Name>
 <Location>EMV</Location>
 </Account>
 </ListOfAccount>
</SiebelMessage>
```

Any child contacts that once belonged to the account will still remain in the database, but will not be associated with this Account.

Execute Method

The Execute method can be specified on the EAI Siebel Adapter to perform combinations of various operations on components in an integration object instance. This method uses the following operations:

- query
- querypage (same as query when used as children operation)
- sync (the same method as Synchronize and is the default operation)
- upsert
- update
- updatesync
- insert
- insertsync
- delete
- skipnode

- skiptree
- none

Note: A none operation is equivalent to operation sync.

These operations perform the same tasks as the related methods. For example, the delete operation makes the EAI Siebel Adapter delete the business component record matched to the particular integration component instance. However, what will be done to the children depends on the combination of the parent operation and the child operation. For information, see *About Execute Method Operations*.

Note: The operation method names are case sensitive. If you misspell an operation method, then the EAI Siebel Adapter assumes the default operation.

An XML document sent to a Siebel application can include operations that describe whether a particular data element must be inserted, updated, deleted, synchronized, and so on. These operations can be specified as an attribute at the component level. They cannot be specified for any other element.

The following XML example demonstrates using the upsert and delete operation to delete a particular child record without updating the parent:

```
<SiebelMessage MessageId="" MessageType="Integration Object" IntObjectName="Sample
Account">
 <ListofSampleAccount>
 <Account operation="upsert">
 <Name>A. K. Parker Distribution</Name>
 <Location>HQ-Distribution</Location>
 <Organization>North American Organization</ Organization>
 <Division/>
 <CurrencyCode>USD</CurrencyCode>
 <Description>This is the key account in the AK Parker Family</
Description>
 <HomePage>www.parker.com</HomePage>
 <LineofBusiness>Manufacturing</LineofBusiness>
 <ListOfContact>
 <Contact operation="delete">
 <FirstName>Stan</FirstName>
 <JobTitle>Senior Mgr of MIS</JobTitle>
 <LastName>Graner</LastName>
 <MiddleName>A</MiddleName>
 <PersonalContact>N</PersonalContact>
 <Account>A. K. Parker Distribution</Account>
 <AccountLocation>HQ-Distribution</AccountLocation>
 </Contact>
 </ListOfContact>
 </Account>
 </ListofSampleAccount>
</SiebelMessage>
```

About Execute Method Operations

Specify an attribute named operation, in lowercase, to the component's XML element. The legal values for this attribute are upsert, sync, delete, query, update, insert, updatesync, insertsync, skipnode, skiptree, and none. If the operation is not specified on the root component, then the sync operation is used as the default.

Note: Specifying an operation within the ListOf tag is not supported. For information on the ListOf tag, see *XML Reference: Siebel Enterprise Application Integration* .

Each child node inherits the operation from the parent if another operation is not explicitly specified. If another operation is explicitly specified, then the following table represents the results of the operation on the current node.

Operation	What Happens to the Current Node	What Happens to Unmatched Children of Current Node
upsert	Update or insert	Leave alone
sync	Update or insert	Delete
update	Update	Delete
updatesync	Update	Delete
insert	Insert	Leave alone
insertsync	Insert	Delete
skipnode	Skip this node	Leave alone
skiptree	Skip the tree	Not applicable

Example of a Parent Node Using a Sync Operation

This example demonstrates the effects of records after a sync operation is performed. The following table is a high level representation of a parent node using the sync operation of the Execute method.

Record In Database	Integration Object Instance	Record After Execute Operation
Account1	Account1 `operation=sync`	Account1
Contact0	Contact1	Contact1
Contact1	Contact2	Contact2

Record in Database

The following code represents GENCOMM0 and GENCOMM1 being retrieved as the contacts for this example:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType =
"Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfAccount>
<Account>
 <AccountStatus>Active</AccountStatus>
 <CurrencyCode>USD</CurrencyCode>
 LanguageCode>ENU</LanguageCode>
 <Location>San Francisco, CA</Location>
```

```
 <Name>GenComm</Name>
 <ListOfContact>
 <Contact>
 <FirstName>GENCOMM0</FirstName>
 <LastName>GENCOMM0</LastName>
 <MiddleName/>
 <Organization>Default Organization</Organization>
 </Contact>
 <Contact>
 <FirstName>GENCOMM1</FirstName>
 <LastName>GENCOMM1</LastName>
 <MiddleName/>
 <Organization>Default Organization</Organization>
 </Contact>
 </ListOfContact>
 /Account>
 </ListOfAccount>
</SiebelMessage>
```

Integration Object Instance

The following code represents the sync operation acting on the contacts from the database.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType =
"Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfAccount>
<Account operation="sync">
 <AccountStatus>Inactive</AccountStatus>
<CurrencyCode>USD</CurrencyCode>
<LanguageCode>ENU</LanguageCode>
<Location>San Francisco, CA</Location>
<Name>GenComm</Name>
<ListOfContact>
 <Contact>
 <FirstName>GENCOMM1</FirstName>
 <LastName>GENCOMM1</LastName>

 <MiddleName/>
 <Organization>Default Organization</Organization>
 </Contact>
 <Contact>
 <FirstName>GENCOMM2</FirstName>
 <LastName>GENCOMM2</LastName>
 <MiddleName/>
 <Organization>Default Organization</Organization>
 </Contact>
</ListOfContact>
</Account>
 </ListOfAccount>
</SiebelMessage>
```

Result Record in Database

The following code represents the results of the sync operation after acting on the two contacts from the database.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType =
"Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfAccount>
<Account>
<AccountStatus>Inactive</AccountStatus>
<CurrencyCode>USD</CurrencyCode>
<LanguageCode>ENU</LanguageCode>
```

```
<Location>San Francisco, CA</Location>
<Name>GenComm</Name>
<ListOfContact>
 <Contact>
 <FirstName>GENCOMM1</FirstName>
 <LastName>GENCOMM1</LastName>
 <MiddleName/>
 <Organization>Default Organization</Organization>
 </Contact>
 <Contact>
 <FirstName>GENCOMM2</FirstName>
 <LastName>GENCOMM2</LastName>
 <MiddleName/>
 <Organization>Default Organization</Organization>
 </Contact>
</ListOfContact>
</Account>
 </ListOfAccount>
</SiebelMessage>
```

In this case, if a matching Account1 exists in the database, then the EAI Siebel Adapter will perform an update of that record. If no record matching Account1 exists, then the EAI Siebel Adapter will insert a new account.

For all the matching child contacts, the sync operation is inherited. Therefore, if the child exists, then it will be updated. If the child does not exist, then it is inserted. Any child contacts that exist in the database but do not match the integration object instance (unmatched children) are deleted.

The reason for this logic is that the sync operation makes the record in the database look like the integration object instance.

Example of a Parent Node Using an Update Operation

This example demonstrates the effects of records after an update operation is performed. the following table is a high level representation of a parent node using the update operation of the Execute method.

> **Note:** The examples represented by the following table, second table in this topic and the fourth table in this topic basically have the same result. However, as reflected in the third table in this topic, the children do not automatically inherit *Update* if it is only set for the root.

Record In Database	Integration Object Instance	Record After Execute Operation
Account1	Account1 `operation=update`	Account1
Contact0	Contact1	Contact1
Contact1	Contact2	Contact2

In this case, if a record matching Account1 exists in the database, then the EAI Siebel Adapter updates that specific record. If no matching account exists, then the result of the EAI Siebel Adapter is an error with this message:

```
Insert operation on integration component 'Account' failed because a matching record
in business component 'Account' with search specification '[Name] = "GenComm" AND
[Location] = "San Francisco, CA"' was found (SBL-EAI-04383)
```

For all the matching child contacts, the update operation is inherited. Therefore, if the child exists, then it will be updated. If the child does not exist, then it is inserted. For child contacts that exist in the database but do not match the

integration object instance, they will be deleted. These might be child contacts created or associated with the Account by default.

This is very similar to the previous example except the record must exist in the database.

Example of a Parent Using an Update Operation and One More Child Using an Insert Operation

This example demonstrates the effects on records after an update operation acts on the parent, and an insert operation acts on one of the children records. The following table is a high level representation of this example.

Record In Database	Integration Object Instance	Record After Execute Operation
Account1	Account1 `operation=update`	Account1
Contact0	Contact1	Contact1
Contact1	Contact2 `operation=insert`	Contact2

In this case, if a record matching Account1 exists in the database, then the EAI Siebel Adapter updates that record. If no record matching Account1 exists, then the result from the EAI Siebel Adapter is an error.

You can also override the parent operation as in the case for Contact2. Since Contact2 does not exist, and there is an explicit insert operation, it will be inserted. Any unmatched children will be deleted as part of the parent operation (update). This is the reason why Contact0 is deleted.

If you are explicitly overriding the parent operation, then you must make sure the operation applies. For example, the two combinations in the following table and the second table in this topic will fail. In the following table, it fails because an insert is attempted when Contact1 already exists in the database.

Record In Database	Integration Object Instance	Record After Execute Operation
Account1	Account1 `operation=update`	None
Contact0	Contact1 `operation=insert`	
Contact1	Contact2 `operation=insert`	

In the following table, the update fails because SubContact3 inherits from Contact2's operation, and Subcontact3 does not exist in the database.

Record In Database	Integration Object Instance	Record After Execute Operation
Account1	Account1	None
Contact1	Contact1	
Contact2	Contact2 `operation=update`	
SubContact1	SubContact1	
SubContact2	SubContact3	

Record In Database	Integration Object Instance	Record After Execute Operation

Example of a Parent Using the Update Operation and One More Child Using the Upsert Operation

This example demonstrates the effects of records after an update operation acts on the parent, and an upsert operation acts on one of the children records. The following table is a high level representation of this example.

Record In Database	Integration Object Instance	Record After Execute Operation
Account1	Account1 `operation=update`	Account1
Contact0	Contact1	Contact1
Contact1	Contact2 `operation=upsert`	Contact2

In this case, if a record matching Account1 exists in the database, then the EAI Siebel Adapter updates that record. If no record matching Account1 exists, then the result of the EAI Siebel Adapter is an error.

For a record matching Contact2, the upsert operation overrides the update operation. Therefore, if Contact2 exists, then it is updated. If no record matching Contact2 is found, then it is inserted. Unmatched child contacts are deleted.

Example of a Parent Using the Upsert Operation and One More Child Using the Sync Operation

This example demonstrates the effects of records after an update operation acts on the parent, and a sync operation acts on one of the children records. The following table is a high level representation of this example.

Record In Database	Integration Object Instance	Record After Execute Operation
Account1	Account1 `operation=upsert`	Account1
Contact0	Contact1	Contact0
Organization2	Organization1	Organization2
Contact1	Contact2 `operation=sync`	Contact1
Organization2	Organization3	Organization1
Contact2		Organization2
Organization2		Contact2
		Organization3

In this case, if a record matching Account1 exists in the database, then the EAI Siebel Adapter updates that record. If no record matching Account1 exists, then the EAI Siebel Adapter inserts the record.

For all child contacts, the upsert operation applies. Therefore, if the child exists, then it is updated. If the child does not exist, then it is inserted. For child contacts that exist in the database, but do not match the integration object instance, they will remain unchanged because upsert does not delete children.

In the case of Contact2, which has the sync operation overriding the upsert operation, it is updated, and its children are synchronized.

Skiptree Operation

The whole sub tree rooted at this node is not processed. It is the same as that whole sub tree not existing in the integration object instance. Operations specified in child nodes do not affect processing in any way since the EAI Siebel Adapter does not act on the child.

```
<?xml version="1.0" encoding="UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId="1-2RE" MessageType="Integration Object"
IntObjectName="Sample Account" IntObjectFormat="Siebel Hierarchical">
 <ListOfSampleAccount>
 <Account operation="upsert">
 <Name>foo </Name>
 <Location>cold storage</Location>
 <ListOfContact>
 <Contact operation="skiptree">
 <FirstName>firstname</FirstName>
 <LastName>contact1</LastName>
 <Organization>Default Organization</Organization>
 <PersonalContact>N</PersonalContact>
 <ListOfBusinessAddress>
 <BusinessAddress operation="insert">
 <City>San Mateo</City>
 <Zip>94402</Zip>
 <AddressName>primary address</AddressName>
 </BusinessAddress>
 </ListOfBusinessAddress>
 </Contact>
 <Contact>
 <FirstName>firstname</FirstName>
 <LastName>contact2</LastName>
 <Organization>Default Organization</Organization>
 <PersonalContact>N</PersonalContact>
 </Contact>
 </ListOfContact>
 </Account>
 </ListOfSampleAccount>
</SiebelMessage>
```

Based on this example, the account is upserted. The processing of the first contact is completely skipped although the business address child has an *insert* operation set. Also, the second contact is upserted.

If the skiptree operation is specified for the account integration component, then the EAI Siebel Adapter skips processing the complete account. This results in no operation. It is possible to have many accounts with some having skiptree specified as shown in the following example. The EAI Siebel Adapter processes the trees that do not have skiptree specified.

```
<?xml version="1.0" encoding="UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId="1-2RE" MessageType="Integration Object"
IntObjectName="Sample Account" IntObjectFormat="Siebel Hierarchical">
 <ListOfSampleAccount>
 <Account operation="skiptree">
 <Name>foo</Name>
 <Location>cold storage<Location/>
```

```
 </Account>
 <Account operation="upsert">
 <Name>bar</Name>
 <Location>cold storage<Location/>
 </Account>
 </ListOfSampleAccount>
</SiebelMessage>
```

Skipnode Operation

Similar to all other Execute operations, the children nodes inherit the semantics of the operation from the parent nodes. If a node has the skipnode operation set, then the EAI Siebel Adapter will skip setting field values for all children unless a child has an explicit operation set that will override.

```
<?xml version="1.0" encoding="UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId="1-2RE" MessageType="Integration Object"
IntObjectName="EAI Account" IntObjectFormat="Siebel Hierarchical">
 <ListOfAccount>
 <Account operation="skipnode">
 <Name>foo</Name>
 <Location>cold storage</Location>
 <ListOfContact>
 <Contact operation="upsert">
 <IntegrationId>1-123</IntegrationId>
 <FirstName>firstname</FirstName>
 <LastName>contact1</LastName>
 <ListOfContact_Organization>
 <Contact_Organization>
  <Organization operation="insert">MyOrganization</Organization>
 </Contact_Organization>
 </ListOfContact_Organization>
 </Contact>
  <Contact operation="upsert">
 <IntegrationId>2-123</IntegrationId>
 <FirstName>firstname</FirstName>
 <LastName>contact2</LastName>
 </Contact>
 </ListOfContact>
 </Account>
 </ListOfAccount>
</SiebelMessage>
```

Based on this example, the account is skipped. However, the EAI Siebel Adapter will attempt to insert the two contacts.

EAI Siebel Adapter Business Service Method Arguments

Each of the EAI Siebel Adapter methods takes arguments that allow you to specify required and optional information to the adapter. You can locate the arguments for each method (and whether it can be used as an input argument, output argument, or both) in the following table.

Argument	Query	Query Page	Sync	Upsert	Update	Insert	Delete	Execute
BusObjCacheSize	Input	Input	Input	Input	Input	Input	Input	Input
DeleteByUserKey	not applicable	not applicable	not applicable	not applicable	not applicable	not applicable	Input	Input
ErrorOnNonExisting Delete	not applicable	not applicable	not applicable	not applicable	not applicable	not applicable	Input	Input

Argument	Query	Query Page	Sync	Upsert	Update	Insert	Delete	Execute
ExecutionMode	Input	Input	not applicable	Input	Input	Input	not applicable	Input
IntObjectName	not applicable	not applicable	not applicable	not applicable	not applicable	not applicable	Input	Input
LastPage	not applicable	Output	not applicable	not applicable	not applicable	not applicable	not applicable	Output
MessageId	Input	Input	Input	Input	Input	Input	Input	Input
NewQuery	not applicable	Input	not applicable	not applicable	not applicable	not applicable	not applicable	Input
NumOutputObjects	Output	Output	Output	Output	Output	Output	Output	Output
OutputIntObject Name	Input	Input	not applicable	not applicable	not applicable	not applicable	not applicable	Input
PageSize	not applicable	Input	not applicable	not applicable	not applicable	not applicable	not applicable	Input
PrimaryRowId	Input	not applicable	Output	Output	Output	Output	Input	Input/ Output
QueryByUserKey	Input	not applicable	not applicable	not applicable	not applicable	not applicable	not applicable	Input
SearchSpec	Input	Input	not applicable	not applicable	not applicable	not applicable	Input	Input
SiebelMessage	Input/ Output	Output	Input/ Output	Input/ Output	Input/ Output	Input/ Output	Input/ Output	Input/ Output
SortSpec	not applicable	Input	not applicable	not applicable	not applicable	not applicable	not applicable	Input
StartRowNum	not applicable	Input	not applicable	not applicable	not applicable	not applicable	not applicable	Input
StatusObject	not applicable	not applicable	Input	Input	Input	Input	Input	Input
ViewMode	Input	Input	Input	Input	Input	Input	Input	Input

The following table presents each argument of the EAI Siebel Adapter business service methods.

Argument	Display Name	Description
BusObjCacheSize	Business Object Cache Size	Default is 5. Maximum number of Business Objects instances cached by the current instance of the EAI Siebel Adapter. If set to zero, then the EAI Siebel Adapter does not use the cache.
DeleteByUserKey	Delete By User Key	A Boolean argument. Forces the EAI Siebel Adapter to use only the user keys to identify a record.
ErrorOnNonExisting Delete	Error On Non Existing Delete	A Boolean argument. Determines whether or not the EAI Siebel Adapter aborts the operation if no match is found.
ExecutionMode	Execution Mode	Used to set the direction of a query on a business component. Valid values are ForwardOnly and Bidirectional. The default is Bidirectional. ForwardOnly is more efficient than Bidirectional, and is recommended in cases where you must process a large number of records in the forward direction only (such as for report generation). For operations that are likely to return more than 10000 records, use ForwardOnly to avoid errors. For more information on executing queries, see the topic on the ExecuteQuery business component method in *Siebel Object Interfaces Reference* .
IntObjectName	Integration Object Name	Name of the integration object to delete.
LastPage	Last Page	Boolean indicating whether or not the last record in the query result set has been returned.
MessageId	Message Id	The MessageId can be used to specify the ID for the generated message. By default, the EAI Siebel Adapter generates a unique ID for each message. However, if you want to use the workflow instance ID, then you can use this argument to specify the ID.
NewQuery	New Query	Default is False. Boolean indicating whether a new query will be executed. If set to True, a new query is executed flushing the cache for that particular integration object.
NumOutputObjects	Number of Output Integration Objects	Number of output integration objects.
OutputIntObjectName	Output Integration Object Name	The name of the integration object that is to be output.
PageSize	Page Size	Default is 10. Indicates the maximum number of integration object instances to be returned.

Argument	Display Name	Description
PrimaryRowId	Object Id	The PrimaryRowId refers to the Id field in the Business Component, Row_Id at the table level. PrimaryRowId is only returned as an output argument if you are passing in one integration object instance. If you are passing multiple integration object instances, then this argument is not returned as an output argument. To obtain the ID field when multiple integration objects are processed, use the StatusObject argument.
QueryByUserKey	Query By Key	A Boolean argument. Forces the EAI Siebel Adapter to use only the user keys to perform a query.
SearchSpec	Search Specification	This argument allows you to specify complex search specifications as free text in a single method argument. For information, see *About Using Language-Independent Code with the EAI Siebel Adapter Business Service*.
SiebelMessage	Siebel Message	The input or the output integration object instance.
SortSpec	Sort Specification	Default is the SortSpec of the underlying business component. This argument allows you to specify complex sort criteria as a free text in a single method argument, using any business component fields and standard Siebel sort syntax. For examples, see *Using Siebel Tools* .
StartRowNum	Starting Row Number	Default is 0 (first page). Indicates the row in the result set for the QueryPage method to start retrieving a page of records.
StatusObject	Status Object	This argument tells the EAI Siebel Adapter whether or not to return a status message.
ViewMode	View Mode	Default is All. Visibility mode to be applied to the Business Object. Valid values are: Manager, Sales Rep, Personal, Organization, Sub-Organization, Group, Catalog, and All. Note that the ViewMode user property on the integration object has priority over the ViewMode method argument.

About the SearchSpec Input Method Argument

The SearchSpec input method argument is applicable to the QueryPage, Query, Delete, and Execute methods. This method argument allows you to specify complex search specifications as free text in a single method argument. Expressions within a single integration component are restricted only by the Siebel Query Language supported by the Object Manager. Integration components and fields are referenced using the following notation:

```
[IntCompName.IntCompFieldName]
```

For example, given an integration object definition with two integration components, Account as the root component and Contact as the child component, the following search specification is allowed:

```
([Account.Site] LIKE "A*" OR [Account.Site] IS NULL) AND [Contact.PhoneNumber] IS
NOT NULL
```

This search specification queries accounts that either have a site that starts with the character A, or do not have a site specified. In addition, for the queried accounts, it queries only those associated contacts who have a phone number.

> **Note:** The operator between fields for a particular integration component instance can be AND unless between the same field. You use the DOT notation to refer to integration components and their fields.

You can include the child integration component in a search specification only if its parent components are also included.

About Multivalue Groups in the EAI Siebel Adapter Business Service

Multivalue groups (MVGs) in the business components are mapped to separate integration components. Such integration components are denoted by setting a user property MVG on the integration component to Y. For information on MVGs, see *Integration Objects*.

An integration component instance that corresponds to a primary MVG is denoted by the attribute IsPrimaryMVG set to Y. This attribute is a hidden integration component field and does not have a corresponding business component field.

Each MVG that appears on the client UI is mapped to a separate integration component. For example, in the Orders Entry - Orders screen, there is an account address, a bill-to address, and a ship-to address. Each of these MVGs needs a separate integration component definition. Each field defined for an integration component (represented by the class CSSEAIIntCompFieldDef) maps to a field in the MVG. For such fields, External Name denotes the name of the business component field as it appears on the master business component, and the user property MVGFieldName denotes the name of the business component field as it appears on the MVG business component.

> **Note:** Setting a primary record in an MVG is supported when the Auto Primary property of the underlying multivalue link is specified as Selected, None, or Default.

Setting a Primary Position for a Contact

You have a contact with multiple contact positions in a Siebel application. None of these positions are marked as the primary position for the contact, and you want to select one of them as the primary position.

To specify a contact position as a primary

1. Create your XML file and insert `<IsPrimaryMVG= 'Y'>` before the contact position you want to identify as the primary position for the contact as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
 <?Siebel-Property-Set EscapeNames="false"?>
- <SiebelMessage MessageId="1-69A" IntObjectFormat="Siebel Hierarchical"
MessageType="Integration Object" IntObjectName="Sample Contact">
- <ListOfSampleContact>
- <Contact>
 <FirstName>Pal888</FirstName>
 <IntegrationId>65454398</IntegrationId>
 <JobTitle>Manager</JobTitle>
 <LastName>John888</LastName>
 <MiddleName />
 <PersonUId>1-Y88H</PersonUId>
 <PersonalContact>N</PersonalContact>
- <ListOfContact_Position>
- <Contact_Position IsPrimaryMVG="Y">
 <EmployeeFirstName>Siebel</EmployeeFirstName>
 <EmployeeLastName>Administrator</EmployeeLastName>
 <Position>Siebel Administrator</Position>
```

```
<RowStatus>N</RowStatus>
<SalesRep>SADMIN</SalesRep>
</Contact_Position>
</ListOfContact_Position>
</Contact>
</ListOfSampleContact>
</SiebelMessage>.
```

2. Use the Upsert or Sync method to update the account.

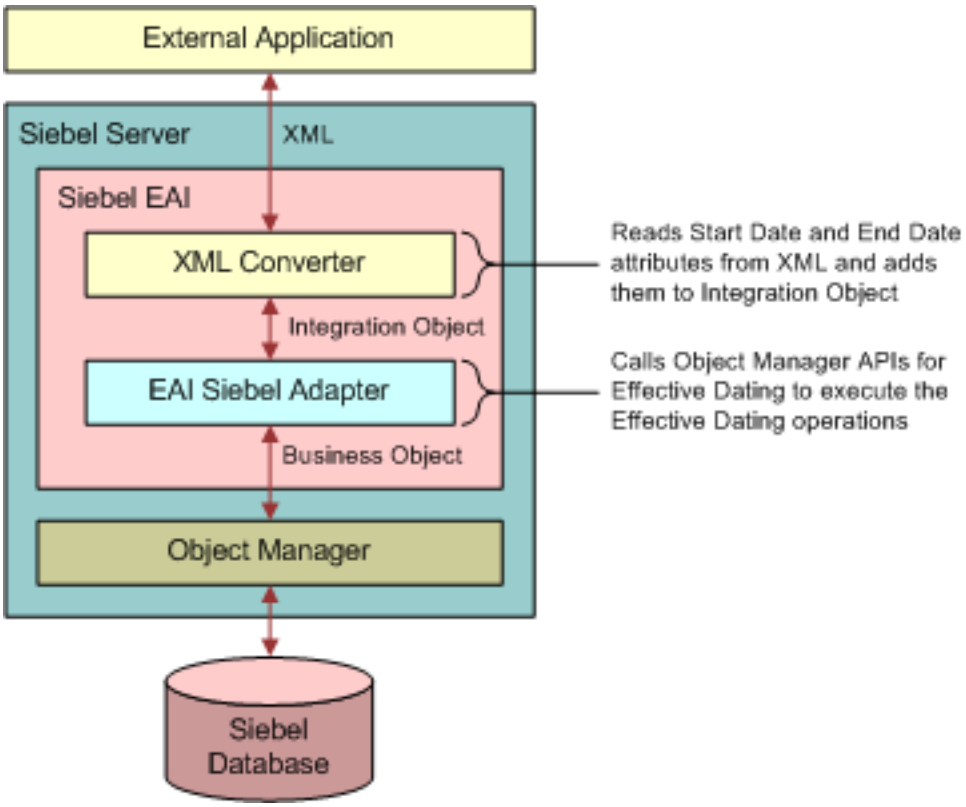
About Using Effective Dating with Siebel EAI Adapter Business Service

The Siebel EAI Adapter allows you to access effective dating data, which means the start date and end date for a given field or link. Third-party applications can request and receive effective dating data from the Siebel application.

To view XML samples for effective dating functionality see *Enabling Effective Dating on Fields*.

You specify effective dating on fields of a given business component through a Siebel Web Client administration screen. For more information, see *Enabling Effective Dating on Fields*. Two integration component fields attributes allow you to set effective dating: EDStartDate and EDEndDate. Standard querying techniques, such as query, insert, update, sync can be used to request effective dating-enabled data.

As the following figure shows, the Siebel Object Manager framework features APIs which are called by the Siebel EAI Adapter when an integration object with effective dating enabled fields or links is read by the XML Converter.



The following two topics explain how effective dating works:

- *Enabling Effective Dating on Fields*
- *Enabling Effective Dating on Links*

Enabling Effective Dating on Fields

This topic explains how effective dating works on fields. It contains the following topics:

- *Configuring Integration Components for Effective Dating on Fields*
- *How the XML Converter Reads Effective Dating Data from Fields*
- *WSDL Schema Generation for Effective Dating on Fields*

Configuring Integration Components for Effective Dating on Fields

You can enable effective dating on a field in an existing integration object if the corresponding field in the business component is effective dating-enabled. You enable fields for effective dating through the Siebel Web Client administration Effective Dating screen.

To enable effective dating on fields

1. In the Siebel Web Client, navigate to Administration - Effective Dating, then Field.
2. In the Effective Dating Buscomp list, select the required business component.

   The list displays the fields already enabled for ED.
3. If you want enable effective dating field not present in this list, click New, then in the Field field click the Select button.
4. In the Business Component Fields window select the required field, then click OK.

   This enables the fields for ED.

You use Siebel Tools to synchronize the object from your current repository, with its underlying business object in the Siebel database which contains the new EDEnabled user property. For more information on synchronizing integration objects, see *Synchronizing Integration Objects*.

To enable effective dating on an integration object

1. In Siebel Tools, select the integration object you want to enable for effective dating.
2. Click the Synchronize button in the Integration Objects list.

   The Integration Object Synchronize wizard appears.
3. Click the plus symbol to display all the related integration components.
4. Uncheck the boxes beside the objects and components you do not want to include in the synchronization of your Siebel integration object.
5. Locate the integration component containing the effective dating-enabled fields and drill down on it.
6. Click the plus symbol to display all the user properties of the field.
7. Locate the EDEnabled user property from the list and add it to the repository side.
8. Review the summary, and if changes are needed, click Back and make the needed changes.

9. If no changes are needed, click Finish to synchronize the Siebel integration object and the Siebel business object.

How the XML Converter Reads Effective Dating Data from Fields

The XML converter reads effective dating attributes contained in an XML file, maps them to a property set, and converts the property set to an integration object instance by embedding the EDStartDate and EDEndDate attributes inside the field name.

For example: If a SOAP request contains the following query for an effective dating field:

```
<acc:EDListOfFirstName>
 <acc:FirstName EDStartDate="04/01/2012" EDEndDate=">James</acc:FirstName>
</acc:EDListOfFirstName>
```

The data will be converted into a child property set of the integration component instance as follows:

```
c[0] CCFPropertySet@1DA79960 p#0 c#1 type="ListOfRelated Contact" vt=0 value="
{
 c[0] CCFPropertySet@1D9FD870 p#1 c#2 type="Related Contact" vt=0 value="
 {
 p["Contact Id"] = "Contact1";
 c[0] CCFPropertySet@1FD92BB0 p#0 c#1 type="EDListOfFirst Name" vt=0 value="
 {
 c[0] CCFPropertySet@13258470 p#2 c#0 type="First Name" vt=3 value="James"
 {
 ["EDEndDate"] = '';
 ["EDStartDate"] = "04/25/2012";
 }
 }
 }
}
```

The dates are now embedded into the field name.

WSDL Schema Generation for Effective Dating on Fields

Effective dating requires two complex type attributes StartDate and EndDate for each effective dating-enabled field. In the following schema example the location field is enabled for effective dating, as shown by the two additional attributes EDStartDate and EDEndDate. Historical data can be retrieved by setting the cardinality of the effective dating-enabled XSD element to *unbounded*.

WSDL Schema Example

```
<xsd:complexType name="RelatedContact">
 <xsd:sequence>
 <xsd:element name="ContactId" maxOccurs="1" minOccurs="0" type="xsd:string"/>
 <xsd:element name="EDListOfFirstName" maxOccurs="1" minOccurs="0"
 type="xsdLocal1:EDListOfFirstName"/>
 <xsd:element name="ContactIntegrationId" maxOccurs="1" minOccurs="0"
 type="xsd:string"/>
 <xsd:element name="EDListOfLastName" maxOccurs="1" minOccurs="0"
 type="xsdLocal1:EDListOfLastName"/>
 . . . .
 </xsd:sequence>
 </xsd:complexType>
```

```
<xsd:complexType name="EDListOfFirstName">
 <xsd:sequence>
 <xsd:element name="FirstName" maxOccurs="unbounded" minOccurs="0"
 type="xsdLocal1:FirstName"/>
 </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="FirstName">
 <xsd:simpleContent>
 <xsd:extension base="xsd:string">
 <xsd:attribute name="EDStartDate" type="xsd:string"/>
 <xsd:attribute name="EDEndDate" type="xsd:string"/>
 </xsd:extension>
 </xsd:simpleContent>
</xsd:complexType>
```

SOAP Query Example

The previous WSDL schema example allows you to generate the following SOAP query:

```
<ElementName EDStartDate=dd1/mm1/yyyy1, EDEndDate = dd2/mm2/yyy2>value</ElementName>
```

For example:

```
<acc:RelatedContact>
 <acc:ContactId>88-30ARL</acc:ContactId>
 <acc:EDListOfFirstName>
 <acc:FirstName EDStartDate="04/01/2012" EDEndDate=">James</acc:FirstName>
 </acc:EDListOfFirstName>
 <acc:EDListOfLastName>
 <acc:LastName EDStartDate="04/01/2012" EDEndDate=">Bond</acc:LastName>
 </acc:EDListOfLastName>
</acc:RelatedContact>
```

Similarly you can perform insert, update, and synchronize operations on data using the WSDL Schema in the previous example.

Enabling Effective Dating on Links

This topic explains how to enable effective dating on links. It contains the following topics:

- *Enabling Effective Dating on Links*
- *Siebel EAI Adapter Operations for Effective Dating on Links*

Enabling Effective Dating on Links

You can enable effective dating on a link in an existing integration object if the corresponding link in the business component is effective dating-enabled. You enable link for effective dating through the Siebel Web Client administration Effective Dating screen.

To enable effective dating on links

1. In the Siebel Web Client, navigate to Administration - Effective Dating, then Child Buscomp.
2. In the Effective Dating Buscomp list, select the required business component, then in the Child Buscomp view select the required link if it is shown.

   If you need to create a new link, see Step 3.

3. Click New, then in the Link Name field click the Select button.
4. In the Link window select the required link, then click OK.

Web Service Schema Example

The following Web service schema example shows a link between the Household and Related Contact business components which have been enabled for effective dating. The effective dating attributes are displayed in bold text.

```
<xsd:complexType name="Household">

 <xsd:complexType name="RelatedContact">
 <xsd:attribute name="EDStartDate" type="xsd:string" />
 <xsd:attribute name="EDEndDate" type="xsd:string" />
<xsd:sequence>
 <xsd:element name="ContactIntegrationId" maxOccurs="1" minOccurs="0"
 type="xsd:string" />
 <xsd:element name="EDListOfFirstName" maxOccurs="1" minOccurs="0"
 type="xsdLocal1:EDListOfFirstName"/>
 <xsd:element name="EDListOfLastName" maxOccurs="1" minOccurs="0"
 type="xsdLocal1:EDListOfLastName"/>
 <xsd:element name="MiddleName" maxOccurs="1" minOccurs="0" type="xsd:string"
 />
 <xsd:element name="PersonUId" maxOccurs="1" minOccurs="0" type="xsd:string" /
 >
 <xsd:element name="PersonalContact" maxOccurs="1" minOccurs="0"
 type="xsd:string" />
 <xsd:element name="ContactId" maxOccurs="1" minOccurs="0" type="xsd:string" /
 >
 <xsd:element name="DateEnteredHousehold" maxOccurs="1" minOccurs="0"
 type="xsd:string" />
 <xsd:element name="DateExitedHousehold" maxOccurs="1" minOccurs="0"
 type="xsd:string" />
 <xsd:element name="PrimaryOrganizationId" maxOccurs="1" minOccurs="0"
 type="xsd:string" />
 <xsd:element name="Relationship" maxOccurs="1" minOccurs="0"
 type="xsd:string" />
 </xsd:sequence>
 <xsd:attribute name="IsPrimaryMVG" type="xsd:string" />
 </xsd:complexType>
</xsd:complexType>
```

This will produce the following XML:

```
<hous:Household>

 <hous:ListOfRelatedContact>
 <hous:RelatedContact EDStartDate="1/1/2003" EDEndDate="">

 <hous:ContactId>88-30KSP</hous:ContactId>

 < hous:EDListOfFirstName>
 < hous:FirstName EDStartDate="04/01/2012" EDEndDate="">SF1N6</acc:James>
 </ hous:EDListOfFirstName>
 < hous:EDListOfLastName>
 < hous:LastName EDStartDate="04/01/2012" EDEndDate=">SL1N6</acc:Bond>
 </ hous:EDListOfLastName>
 < hous:MiddleName>MN1</hous:MiddleName>

 </hous:RelatedContact>

 </hous:ListOfRelatedContact>
 </hous:Household>
```

Siebel EAI Adapter Operations for Effective Dating on Links

The Siebel EAI Adapter receives the integration object in the format described in the *Enabling Effective Dating on Links* . Depending on the specified operations the effective dates are used as described in the following:

Insert operation

Once the record is inserted into the parent and child business component, the Siebel EAI Adapter reads the EDStartDate and the EDEndDate from the integration object and inserts these values into the corresponding effective dating business component.

Update operation

Once the record is inserted into the parent and child business component, the Siebel EAI Adapter removes all history records from the effective dating enabled business component and then reads the EDStartDate and EDEndDate values from the integration object and inserts these dates as fresh records into the business component.

> **Note:** The Update operation is only possible for currently active links (in other words, update is not possible if a link has been soft deleted by giving a end date value for most recent history record).

Upsert and Synchronize Operations

If the upsert attribute is specified for the operation in the integration component then no history records are deleted, instead the history given in the XML input is inserted into the history table.

The synchronize operation can result in the insert, update or deletion of the child integration component as specified in the conditions set in the *Synchronize Method*.

XML Example

This example illustrates what is required if you want to perform an insert or upsert operation to insert or update multiple history records for the same child business component.

> **Note:** Multiple entries must be specified in the input XML code with same user key.

```
<hous:Household>
 <hous:ListOfRelatedContact>
 <hous:RelatedContact EDStartDate="1/1/2003" EDEndDate="">
 <hous:ContactId>88-30KSP</hous:ContactId>
 </hous:RelatedContact>
 <hous:RelatedContact EDStartDate="1/1/2002"
 EDEndDate="12/31/2002">
 <hous:ContactId>88-30KSP</hous:ContactId></hous:RelatedContact>
 <hous:RelatedContact EDStartDate="1/1/2001"
 EDEndDate="12/31/2001">
 <hous:ContactId>88-30KSP</hous:ContactId>
 </hous:RelatedContact>
 <hous:RelatedContact EDStartDate="1/1/2000"
 EDEndDate="12/31/2000">
 <hous:ContactId>88-30KSP</hous:ContactId>
 </hous:RelatedContact>
 <hous:ListOfRelatedContact>
 </hous:Household>
```

About Using Language-Independent Code with the EAI Siebel Adapter Business Service

If the user property AllLangIndependentVals is set to Y at the integration object level, then the EAI Siebel Adapter uses the language-independent code for its LOVs.

In the outbound direction, for example, the Query method, if the AllLangIndependentVals is set to Y, then the EAI Siebel Adapter translates the language-dependent values in the Siebel Database to their language-independent counterpart based on the List Of Values entries in the database.

In the inbound direction, for example the Synchronize method, if the AllLangIndependentVals is set to Y, then the EAI Siebel Adapter expects language-independent values in the input message, and translates them to language-dependent values based on the current language setting and the entries in the List Of Values in the database.

Note: The LOV-based fields are always validated using language-dependent values. Using language independent values for LOVs and MLOVs increases the EAI Siebel Adapter CPU usage by about five percent, but allows easier communication between systems that operate on different languages.

About LOV Translation and the EAI Siebel Adapter Business Service

The Siebel application distinguishes two types of lists of values (LOV):

- **Multilingual LOV (MLOV).** Stores a language-independent code (LIC) in the Siebel Database that is translated to a language-dependent value (LDV) for active language by Object Manager. MLOVs are distinguished by having the Translation Table specified in the Column definition.
- **Single-language LOV.** Stores the LDV for the current language in the Siebel Database. The Boolean integration object user property AllLangIndependentVals determines whether the EAI Siebel Adapter will use LDV (No = no translation necessary) or LIC (Yes = translation needed) for such LOVs.

Translating to LIC affects performance, but allows easier cooperation between systems that operate on different languages. This option is especially used by various import and export utilities.

The AllLangIndependentVals integration object user property is undefined for integration objects when the base object type is not Siebel Business Object. When the base object is Siebel Business Object, AllLangIndependentVals is defined with a default value of N.

The following table explains the behavior of the EAI Siebel Adapter according to the AllLangIndependentVals integration object user property values.

AllLangIndependentVals	Yes	No	Undefined
LOV	LIC	LDV	LDV
MLOV	LIC	LDV	LIC

Siebel EAI and Run-Time Events

The Siebel application allows triggering workflows based on run-time events or workflow policies.

Run-Time Events. Siebel EAI supports triggering workflows based on run-time events such as Write Record, which is triggered whenever a record is written. If you use the EAI Siebel Adapter to import data into Siebel CRM, and use run-time events, then consider the following:

For the EAI Siebel Adapter, one call to the EAI Siebel Adapter with an input message is a transaction. Within a transaction, the EAI Siebel Adapter makes multiple Write Record calls. At any point in the transaction, if the EAI Siebel Adapter encounters a problem the transaction is rolled back entirely. However, if you have specified events to trigger at Write Record, such events are called as soon as the EAI Siebel Adapter makes Write Record calls even though the EAI Siebel Adapter might be in the middle of a transaction. If you have export data workflows triggered on such events, this might lead to exporting data from Siebel CRM that is not committed and might be rolled back. It is also possible that your events are triggered when the record is not completely populated, which leads to situations that are not handled by your specified event processing workflow.

To avoid the effects of this interaction between the EAI Siebel Adapter and run-time events use the business service EAI Transaction Service to figure out if a transaction (typically, the EAI Siebel Adapter) is in progress. You might then want to skip processing that is not desirable when the EAI Siebel Adapter is in progress.

For example, suppose you have a workflow to export orders from Siebel CRM, which is triggered whenever the order record is written. You also import orders into Siebel CRM using EAI. In such a situation, you do not want to export orders while they are being imported, because the import might be aborted and rolled back. You achieve this using the EAI Transaction Service business service as the first step of the export workflow. If you find that a transaction is in process you can branch directly to the end step.

Workflow Policies. In addition to Run-Time Events, Siebel CRM also supports Workflow Policies as a triggering mechanism for workflows. You can use workflow policies instead of run-time events to avoid the situation discussed in this topic. Use Workflow Policies instead of Run-Time Events when possible.

Guidelines for Using the EAI Siebel Adapter Business Service

The following guidelines are to be considered when using the EAI Siebel Adapter:

- Keep the integration objects small. Basically, inactivate any unused fields in the integration component. Avoid creating large integration object instances.
- Test the developed object definitions using the EAI Siebel Adapter before adding to production. You must test your input and output using working and negative scenarios. Also do performance testing to make sure you are satisfied with the performance of the input and the output.
- Oracle does not support the use of EAI to update data that is based on administration-type business components such as Client - Mobile or Position. Only the System Administrator updates these types of data.
- Always use a search specification with the Query() method to avoid receiving every object when run.
- To optimize database performance, you can explicitly specify that the EAI Siebel Adapter use only user key fields. This feature is available for the methods Query, Delete, and Execute. To use it, set the input property

QueryByUserKey to True for the EAI Siebel Adapter business service and pass an integration object instance (for example, a Siebel Message) as an input as well. By default, the Siebel adapter uses all the fields in the input integration object instance.

Troubleshooting the EAI Siebel Adapter Business Service

The EAI Siebel Adapter natively accesses Siebel objects definitions using the business objects, integration objects, and business component classes. Because of this design, you might get an EAI Siebel Adapter error that contains an error message from the Siebel Object Manager. See the figure in the topic *About the Difference Between Integration Objects and Integration Object Instances* for a logical overview of the Siebel architectural layers. The figure in the topic *About the Difference Between Integration Objects and Integration Object Instances* also shows the component events that will help you determine in which layer of the application the problem is occurring.

The EAI Siebel Adapter functionality must be considered in light of the entire application functionality. For example, the Siebel Communications product line provides preconfigured Asset Based Ordering functionality that uses Siebel workflows and business services. The workflows use the EAI Siebel Adapter business service to extract data from the database and to update the database.

When using this functionality, the possibility exists that you might get an error in a step of the workflow that indicates a problem with the EAI Siebel Adapter, such as the asset you want to insert already exists in the system. In this case, first verify that you are not inserting a duplicate asset. If you have validated that the asset is new and not a duplicate, then you must research the specifics as to why the EAI Siebel Adapter failed to insert the new asset or attempted to insert a duplicate asset.

If you have modified the preconfigured Asset integration object or business object, it could be one of your customizations. For example, perhaps your asset requires additional fields, and you are not providing those fields in your inbound integration object instance. Therefore, it uses any default values, thus creating a potential duplicate asset.

Enabling Logging for the EAI Siebel Adapter Business Service

Using component events, logging can be done in the Siebel application. Components are used to assist with the debugging of problems in the Siebel application. A list of useful and relevant component events for debugging EAI Siebel Adapter problems are listed in the following table. These components events can be enabled on any server component that is capable of running an EAI process and on the Siebel client. You might want to enable other events not listed in the following table.

Event Alias Name	Logging Level	Description
EAISiebAdpt	4 or 5	Captures EAI Siebel Adapter related events, including integration component and integration component fields accessed and the values for the fields; business components and business component fields accessed and the values for the fields. This is the main event to enable for EAI Siebel Adapter troubleshooting.

Event Alias Name	Logging Level	Description
EAISiebAdptPerf	4	Captures EAI Siebel Adapter performance related events, including operation performed and time for the operation in milliseconds. This event summarizes the result of the EAI Siebel Adapter operation. For more information on performance logging, see *Troubleshooting the EAI Siebel Adapter Business Service* and Doc ID 476905.1 on My Oracle Support. This document was previously published as Siebel FAQ 1840.
EAISiebAdptSvcArgTrc	3 or 4	Dumps the inputs and output arguments for the EAI Siebel Adapter when EnableServiceArgTracing=true. For more information on argument tracing, see *Enabling Siebel Argument Tracing*.
EAITransaction	4	Captures when an EAI Transaction starts.
EAIInfra	4	Output Message: IntObjType=Contact Interface Format=Siebel Hierarchical
EAIQrySpec	4	Captures the search specification if one is specified.
SQL	4	Captures SQL executed against the database.
SQLParseAndExecute	4	Captures SQL statements and shows SQL bind parameters executed. Shows SQL executed against the database. Might sometimes be different than the SQL show in ObjMgrSQLLog.
ObjMgrLog	4 or 5	Logs error code and error message encountered by various Siebel objects.
ObjMgrDataLog	4	Logs the beginning of a transaction for the database connection.
ObjMgrBusServiceLog	4	Captures creation, deletion and invocation of a Business Service.
ObjMgrBusCompLog 4	4 or 5	Captures the beginning and end of the Business Component creation and deletion.

For all the events listed in the previous table, setting the logging level to level 4 is sufficient for most types of testing. You can set the component event to level 5 if you want to see debug level output, but it is not generally recommended as it adds more lines of data to the log file that might or might not be helpful. Logging level 4/5 represents that a logging level of 4 or 5 is supported.

To enable EAI Siebel Adapter logging

1. Navigate to the Administration - Server Configuration screen, Servers view.
2. In the starting applet, select the Siebel Server that you want to enable EAI Siebel Adapter logging.

3. In the middle applet, select the Components tab, and highlight the component.
4. In the lower applet, select the Events tab, and set component events.

When you enable the component event logging, make sure you select the appropriate server component or components involved in the process. For example, if you are testing receiving XML data in the MQSeries Server Receiver, then you would enable logging on the MQSeriesSrvRcvr component.

You can also use the same srvrmgr command to turn on the component event logging. You will use the `"%"` shortcut syntax to enable events. An example of this syntax is **`"change evtloglvl EAISIEB%=4 for comp BusIntMgr".`**

Enabling Siebel Argument Tracing

You can also export input and output arguments in XML format to a file for the EAI Siebel Adapter. These XML files represent the input and output arguments integration object instances. This is a useful technique as it writes to a file the integration object instances in the directory where your Siebel process is running. For example, in the Siebel Developer Web Client, it might be **`c:/siebel/bin`**.

To enable output arguments tracing

1. Set the server parameter EnableServiceArgTracing to True:
   - If you are running the Siebel Developer Web Client, then add the following to your .cfg file:

     ```
     [EAISubsys]

     EnableServiceArgTracing = TRUE
     ```

   - If you are running the Siebel Web Client, modify the following Siebel Server parameter for your object manager:

     ```
     "EnableServiceArgTracing" = true
     ```

2. Set the appropriate component event level on your server component through the server manager on the server or SIEBEL_LOG_EVENTS in the Siebel Developer Web Client.

   If you set event to:

   =3, then input arguments will be written out to a file when an error happens.

   =4, then input and output arguments will be written to a file.

Configuring the EAI Siebel Adapter Business Service for Concurrency Control

The EAI Siebel Adapter supports concurrency control to guarantee data integrity and avoid overriding data by simultaneous users or integration processes. To do so, the EAI Siebel Adapter uses the Integration Component Key called the Modification Key.

Modification Key

A Modification Key is an Integration Component Key of the type *Modification Key*. A Modification Key is a collection of fields that together are used to verify the version of an integration component instance. Typically, Modification Key fields are Mod Id fields for the tables used. Multiple Modification Key fields might be needed, because a business component might be updating multiple tables, either as extension tables, or through implicit or explicit joins.

The EAI Siebel Adapter methods (Insert, Update, Synchronize, Upsert) check for the existence of a Modification Key. If no Modification Key is specified in the integration component definition, or if Modification Key fields are not included in the XML request, then the EAI Siebel Adapter does not check for the record version and proceeds with the requested operation. If a valid Modification Key is found, but the corresponding record cannot be found, then the EAI Siebel Adapter assumes that the record has been deleted by other users and returns the error SSASqlErrWriteConflict.

If a valid Modification Key as well as the corresponding record can be found, then the EAI Siebel Adapter checks if the Modification Key fields in the XML request and the matched record are consistent. If any of the fields are inconsistent, then the EAI Siebel Adapter assumes that the record has been modified by other users and returns the error SSASqlErrWriteConflict. If all the fields are consistent, then the EAI Siebel Adapter proceeds with the requested operation.

Modification IDs

To determine which Mod Id fields must be used as part of a Modification Key, you expose Mod Id fields for tables whose columns might be updated by that integration object. In some situations you might have to add corresponding integration component fields as well as business component fields.

> **Note:** The EAI Siebel Adapter can update base and extension tables. It might even update joined table columns through picklists that allow updates.

When using Modification IDs, the following behaviors are present:

- All fields must be present in the integration object instance for the Mod Key to be used.
- Only one defined Modification Key is present for each integration component. Unlike for User Keys, multiple Mod Keys are not allowed.

About the Modification ID for a Base Table

The integration component field Mod Id for a base table is created by the Integration Object Builder Wizard, but you must make sure it is active if it is needed for your business processes.

About the Modification ID for an Extension Table

An extension table's Mod Id field is accessible as extension table name.Mod Id in the business component, for example, S_ORG_EXT_X.Mod Id. However, if your business processes require this field, then you must manually add it to the integration object definition by copying the Mod Id field and changing the properties.

About the Modification ID for a Joined Table

A joined table's Mod Id field must be manually added in both business component and integration object definitions. Business component Mod Id fields for joined tables must:

- Be prefixed with CX string and preferably followed by the name of the join

- Be Joined over the correct join
- Have MODIFICATION_NUM specified as underlying column of type DTYPE_INTEGER

About MVG and MVGAssociation Integration Components

For integration components that are of type MVG or MVGAssociation, in addition to the preceding steps, you must create user properties MVGFieldName and AssocFieldName for each Modification ID integration component field, respectively, and set the name of the field shown in the parent business component as the value.

To configure the EAI Siebel Adapter business service for concurrency control

1. For each integration component, identify all needed Modification IDs:

   > **Note:** In addition to the Modification ID for the base table, Modification IDs for tables that are used through one-to-one extension as well as through implicit joins are relevant. For example, on modifying an account record MODIFICATION_NUM column on S_ORG_EXT is updated, not the MODIFICATION_NUM column on S_PARTY.

   a. Identify all active fields in an integration component that will be updated and have to be concurrency safe.
   b. Select the corresponding business component, the value in the External Name property of the integration component.
   c. For each field identified in Step a, check the value of the Join property of the field. If the join is not specified, then the field belongs to the base table; otherwise, note the name of the join.
   d. In the Object Explorer, select Business Component, then Join, and query for the business component from Step b. Search whether there is an entry whose Alias property matches the name of the join from Step c:

      - If a matching Alias is found, then this field belongs to a Joined Table. The name of the join in Step c is the join name, and the value of the Table property is the joined table.
      - If no Alias matches, then this is an implicit join to an Extension Table. The name of the join in Step cis the name of the extension table.

2. Create business component fields for Mod Ids of Joined Tables. For the previous example, create a new field in the business component Account with the following settings:

   - **Name.** CX_Primary Organization-S_BU.Mod Id
   - **Join.** Primary Organization-S_BU
   - **Column.** MODIFICATION_NUM
   - **Type.** DTYPE_INTEGER

3. Expose all Modification IDs identified in Step 1 as integration component fields.
4. For MVG and MVG Association integration components, add user property MVGFieldName and AssocFieldName respectively, on all Modification ID fields as follows:

   a. Check the Integration Component User Prop sub type for user properties of the integration component.
   b. If there is a user property called MVGAssociation, then the integration component is a MVG Association, but if there is a user property called Association then the integration component is a MVG.

      > **Note:** If the integration component is neither an MVG nor an MVG Association, then nothing is required to be done.

5. Repeat the following steps for each Modification ID field on the integration component:

a. Add user property MVGFieldName if MVG, or AssocFieldName if MVG Association.
b. Set the value of the user property to the same as the field name, for example, Mod Id, *extension table name*.Mod Id, or CX_*join*.Mod Id.

6. Create Modification Key.

   Define a new integration component key of type Modification Key, and include all the integration component fields exposed in Step 3 to this key.
7. Validate integration objects and deliver the workspace.
8. Modify client program to use the Modification Key mechanism:

   a. The client program must store the value of the Modification IDs when it queries data from the Siebel Database.
   b. The client program must send exactly the same values of the Modification IDs that it retrieved from the Siebel Database when sending an update.
   c. The client program must not send any Modification IDs when sending a new record to the Siebel application. If this is violated, then the client program generates an error indicating that the record has been deleted by another user.

Integration Component Account Example

Consider an integration component Account of the business component Account:

- Field Home Page has property Join set to S_ORG_EXT. This is an implicit join, because it is not listed in the joins; therefore, this field belongs to Extension Table S_ORG_EXT.
- Field Primary Organization has property Join set to Primary Organization-S_BU. This is an explicit join, because it is listed in the joins. The value of Table property is S_BU; therefore, this field belongs to Joined Table S_BU joined over Primary Organization-S_BU.
- Activate integration component field Mod Id:
  - Set Name, External Name, XML Tag properties to Mod Id
  - Set External Data Type property to DTYPE_NUMBER
  - Set External Length property to 30
  - Set Type property to System
- Add integration component field S_ORG_EXT.Mod Id:
  - Set Name, External Name, XML Tag properties to S_ORG_EXT.Mod Id
  - Set External Data Type property to DTYPE_NUMBER
  - Set External Length property to 30
  - Set Type property to System
- Add integration component field CX_Primary Organization-S_BU.Mod Id:
  - Set Name, External Name, XML Tag properties to CX_Primary Organization-S_BU.Mod Id
  - Set External Data Type property to DTYPE_NUMBER
  - Set External Length property to 30
  - Set Type property to System

Integration Component Account_Organization Example

Consider the integration component Account_Organization of the Sample Account integration object. Account_Organization is an MVG Association as denoted by the presence of the user property MVGAssociation. Assume two Modification IDs, Mod Id and S_ORG_EXT.Mod Id, were exposed on this integration component:

- For field Mod Id create a new user property with the name of AssocFieldName with a value of Mod Id.
- For field S_ORG_EXT.Mod Id create a new user property with the name of AssocFieldName with a value of S_ORG_EXT.Mod Id.

In this integration component example, Account of the Sample Account integration object, takes the following action:

- Create a new Integration Component key called Modification Key.
- Set the type of the key as Modification Key.
- Add integration component fields Mod Id, S_ORG_EXT.Mod Id, and S_BU.Mod Id to the Modification Key.

Status IDs

When using Status IDs with Modification IDs, the following behavior can be present:

- All fields must be present in the integration object instance for the Modification Key to be used.
- Only one defined Modification Key is present for each integration component. Unlike User Keys, multiple Modification Keys are not used with Status IDs.

7 EAI UI Data Adapter Business Service

EAI UI Data Adapter Business Service

The EAI UI Data Adapter business service is used for exposing Siebel data to external user interfaces. This chapter includes the following topics:

- *About the EAI UI Data Adapter Business Service*
- *EAI UI Data Adapter Business Service Methods*
- *EAI UI Data Adapter Business Service Method Arguments*

About the EAI UI Data Adapter Business Service

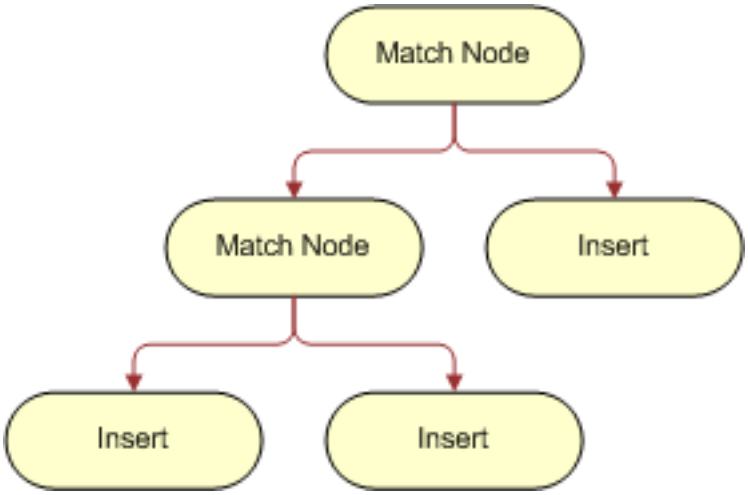
The EAI UI Data Adapter business service exposes an interface with weakly typed arguments that can query and update data in the Siebel database. The EAI UI Data Adapter service is called indirectly by UI Data Sync Services, which are published externally as Web services.

The EAI UI Data Adapter is similar to the EAI Siebel Adapter business service, but contains key differences that make it more suitable for UI rendering by custom Web applications. The differences are summarized as follows:

- **Row Id as User Key.** Unlike the EAI Siebel Adapter, the EAI UI Data Adapter does not use user keys defined in the integration object. It uses an implicit, hard-coded user key, which comprises the Row Id field.

  For more information about how User Keys are used with the EAI Siebel Adapter, see *About Integration Component Keys*.
- **Row Id and Mod Id as Status Key.** Unlike the EAI Siebel Adapter, the EAI UI Data Adapter does not use status keys defined in the integration object. It uses an implicit, hard-coded status key, which comprises the fields Row Id and Mod Id.

  For more information about how Status Keys are used with the EAI Siebel Adapter, see *About Integration Component Keys*.
- **Operation Semantics on Leaf Nodes.** In an integration object hierarchy, nodes with at least one child are called internal nodes and nodes without children are called leaf nodes. When either the insert or update method is called on the EAI Siebel Adapter, the adapter performs the operation on both internal nodes and leaf nodes. When the insert or update method is called on the EAI UI Data Adapter, the adapter performs insert on leaf nodes only as represented in the following figure.



The match nodes in the following figure reflects that the database contains a record with the same user keys as the integration object instance.

- **Predefined Queries.** The EAI UI Data Adapter extends the Query Page functionality of the EAI Siebel Adapter. The EAI UI Data Adapter can take the name of a predefined query and execute the query.
- **Child Pagination.** The EAI UI Data Adapter supports child pagination, enabling custom UIs to render one page of data at a time.

  For detailed information about the QueryPage method, see *QueryPage Method*.

  For more information, see *Root and Child Pagination*.
- **Strongly Typed Data.** Unlike the EAI Siebel Adapter, the EAI UI Data Adapter supports the exchange of strongly typed data.

The EAI UI Data Adapter is most suitable for use in custom UI development where the service is called indirectly by Web services. In other types of integration scenarios, the EAI Siebel Adapter is a more suitable choice. For more information about the EAI Siebel Adapter, see *EAI Siebel Adapter Business Service*.

EAI UI Data Adapter Business Service Methods

The EAI UI Data Adapter service provides access to the following methods:

- *QueryPage Method*
- *UpdateLeaves Method*
- *InitLeaves Method*
- *InsertLeaves Method*
- *DeleteLeaves Method*
- *Execute Method*

QueryPage Method

Custom UIs can use the QueryPage method to query data in the Siebel database one page at a time. QueryPage supports both query-by-example (QBE) and predefined queries (PDQ). However, it is recommended that you use either QBE or a PDQ, but not both at the same time. If both QBE and PDQ are specified, then PDQ overrides QBE. In this case, the EAI UI Data Adapter executes the PDQ, ignores the QBE, and does not generate an error.

QueryPage Method Arguments

The following table lists the method arguments used with the QueryPage method. For a description of the arguments, see *EAI UI Data Adapter Business Service Method Arguments*.

Method Argument Name	Type
ExecutionMode	Input
LOVLanguageMode	Input
NamedSearchSpec	Input
NewQuery	Input
NumOutputObjects	Output
OutputIntObjectName	Input
SiebelMessage	Input / Output
ViewMode	Input

Root and Child Pagination

The EAI UI Data Adapter supports pagination for both root and child components. To support root and child pagination, the EAI UI Data Adapter requires that you set the attributes listed in the following table as part of the integration component instance.

> **Note:** Pagination over root components benefits performance because, as long as the search specification, sort specification, and view mode remain the same, the business component is not re-executed with each invocation of QueryPage. However, for pagination over child components, the component is reexecuted every time you call QueryPage.

Attribute	Description
pagesize	The number of records to be returned for a component. The default page size is 10. Note that there is a server parameter that controls the maximum page size (MaximumPageSize). If the pagesize attribute is greater than the maximum pagesize defined in the server parameter, then an error occurs.
startrownum	Determines the starting point for record retrieval. The 0-based index of the record within the recordset.
lastpage	Indicates whether the record being returned is the last record in the record set. The value is set by the EAI UI Data Adapter. Valid values are true or false.
recordcountneeded	When set to true, indicates that a record count is needed for this component. Valid values are true or false.
recordcount	Value set by the EAI UI Data Adapter indicating the approximate record count provided by the object manager based on the search specification.
child pagination	When set to true, enables pagination of child records. Valid values are true or false.

Example of QueryPage on Parent and Child Components

This example demonstrates querying on both parent and child components. In this example, the query is for accounts that begin with 'A' and any associated contacts (First Name and Last Name). Note that pagesize is 10 and an approximate record count is requested and returned in the response.

Request

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
   <ListOfAccount pagesize="10" startrownum="0" recordcountneeded = "true">
      <Account>
         <Name>='A'</Name>
            <ListOfContact>
            <Contact>
               <FirstName></FirstName>
               <LastName></LastName>
            </Contact>
         </ListOfContact>
      </Account>
   </ListOfAccount>
</SiebelMessage>
```

Response

```
SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
   <ListOfAccount recordcount="2" lastpage="true">
      <Account>
         <Name>Adams Tech</Name>
         <ListOfContact lastpage="true">
         <Contact>
            <FirstName>Sally</FirstName>
            <LastName>Brown</LastName>
         </Contact>
            <Contact>
               <FirstName>Terry</FirstName>
               <LastName>Smith</LastName>
```

```
            </Contact>
         </ListOfContact>
      </Account>
      <Account>
         <Name>Aleph Inc.</Name>
            <ListOfContact lastpage="true">
            <Contact>
               <FirstName>Bill</FirstName>
               <LastName>Jones</LastName>
            <Contact>
            <Contact>
               <FirstName>Roland</FirstName>
               <LastName>Smith</LastName>
            </Contact>
         </ListOfContact>
      </Account>
   </ListOfAccount>
</SiebelMessage>
```

Example of QueryPage using Child Pagination

This example demonstrates querying using child pagination. In this example, the query is for account with name as ABC Mart #18 and any associated contacts (First Name and Last Name). Note that only 10 records are retrieved though there are 4999 records. This is because the page size is 10 and child pagination parameters is also set.

Request

```
<SiebelMessage MessageId="" IntObjectName="EAI Account" MessageType="Integration
Object" IntObjectFormat="Siebel Hierarchical">
     <ListOfEAI_spcAccount>
        <Account Name="ABC Mart #18">
            <ListOfContact recordcountneeded="true" startrownum="0"
ChildPagination="true" pagesize="10">
             <Contact>
             </Contact>
            </ListOfContact>
         </Account>
     <ListOfEAI_spcAccount>
</SiebelMessage>
```

Response

```
<SiebelMessage MessageId="" MessageType="Integration Object" IntObjectName="EAI
Account" IntObjectFormat="Siebel Hierarchical">
     <ListOfEAI_spcAccount lastpage="true">
        <Account Name="ABC Mart #18">
           <ListOfContact recordcount="4999" lastpage="false">
              <Contact First_spcName="M*" Last_spcName="A*"></Contact>
              <Contact First_spcName="MAYA" Last_spcName="ANDERSON"></Contact>
              <Contact First_spcName="ABS_ADMIN_EMP1"
Last_spcName="ABS_ADMIN_EMP1"></Contact>
              <Contact First_spcName="ABS_ADMIN_EMP2"
Last_spcName="ABS_ADMIN_EMP2"></Contact>
              <Contact First_spcName="ABS_ADMIN_EMP3"
Last_spcName="ABS_ADMIN_EMP3"></Contact>
             <Contact First_spcName="ABS_ADMIN_EMP4"
Last_spcName="ABS_ADMIN_EMP4"></Contact>
<Contact First_spcName="HARRY" Last_spcName="ADAMS"></Contact>
              <Contact First_spcName="VERNON" Last_spcName="AJAX" ></Contact>
              <Contact First_spcName="THOMAS" Last_spcName="ALEX" ></Contact>
              <Contact First_spcName="MAY" Last_spcName="ALLISON" ></Contact>
           </ListOfContact>
        </Account>
     </ListOfEAI_spcAccount>
```

```
</SiebelMessage>
```

Sort Specification

You can specify a sort specification on one or more integration component fields of an integration component. For each field you want sort on, you must define the attributes listed in the following table. If both attributes are not specified, then the field is not considered as a sort field.

Attribute	Description
sortorder	Determines whether the sort order is ascending or descending. Valid values are ASC or DEC.
sortsequence	Determines the order in which the sort specification is applied. Valid values are integer numbers.

Example of Sort Specification

This example demonstrates using the QueryPage method with an ascending sort order.

Request

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
   <ListOfAccount>
      <Account>
         <Row_Id>2-1111</Row_Id>
         <ListOfContact pagesize="40" startrownum="0" recordcountneeded="true">
            <Contact>
               <FirstName sortorder="ASC" sortsequence="1"></FirstName>
            </Contact>
         </ListOfContact>
      </Account>
   </ListOfAccount>
</SiebelMessage>
```

Response

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
   <ListOfAccount lastpage="true">
      <Account>
         <Row_ID>2-1111</Row_ID>
         <ListOfContact recordcount="3" lastpage="true">
            <Contact>
               <FirstName>Alice</FirstName>
            </Contact>
            <Contact>
               <FirstName>Bill</FirstName>
            </Contact>
            <Contact>
               <FirstName>Casey</FirstName>
            </Contact>
         </ListOfContact>
      </Account>
   </ListOfAccount>
</SiebelMessage>
```

Predefined Query

You can specify the name of a PDQ using the method argument `NamedSearchSpec`. The EAI UI Data Adapter uses this value to set the search specification at the business object level.

Search Specification

You can use the searchspec attribute on a component instance for complicated queries.

For example, query by example (QBE) uses AND as the implicit operator between fields. You could implement OR semantics by using multiple integration component instances, but this would result in a query for each integration component instance and might result in duplicate records being returned. Using the searchspec attribute could avoid this problem.

The syntax for the searchspec attribute is as follows:

- Expression: *Expression* [*Binary Operator Expression*]
- Expression: [*Field XML tag*] *Operator* '*Value*'
- Expression: (*Expression*)

  > **Note:** Parentheses can be nested.

- Expression: [*Field XML tag*] IS NULL | [*Field XML tag*] IS NOT NULL
- Expression: EXISTS(*Expression*) | NOT EXISTS(*Expression*)

  > **Note:** In EXISTS and NOT EXISTS expressions, use the business component field names of multivalue group (MVG) fields, not the integration component XML tag names.

- Operator: = | ~= | < | <= | > | >= | <> | LIKE | ~LIKE
- Binary Operator: AND | OR

The EAI UI Data Adapter parses the searchspec (unlike the EAI Siebel Adapter) and performs the following operations before setting the search specification on the business component:

- Converts Field XML tags into business component field names. For example, assume two business component fields, First Name and Last Name, have XML tags FirstName and LastName respectively. The EAI UI Data Adapter converts the XML tags as shown in the following table.

This Search Spec	Will be converted to this
`[FirstName] LIKE '*Jon*' AND [LastName] = 'Doe'`	`[First Name] LIKE '*Jon*' AND [Last Name] = 'Doe'`
`[FirstName] LIKE '*Jon*' OR [LastName] LIKE 'Doe*'`	`[First Name] LIKE '*Jon*' OR [Last Name] LIKE 'Doe'`

- If the input argument LOVLanguageMode is set to LIC, then LOV values are converted to language dependent codes. See *EAI UI Data Adapter Business Service Method Arguments*.
- Validates operators, binary operators, and the syntax of the searchspec.

For more information about query language, see *Siebel Developer's Reference* .

Example of Using the searchspec Attribute

This example demonstrates using the searchspec attribute for the QueryPage method.

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
   <ListOfAccount>
      <Account>
         <Id>2-1111</Id>
         <ListOfContact pagesize="10" startrownum="0">
            <Contact searchspec="[FirstName] LIKE '*Jon*' AND [LastName] = 'Doe'">
               <FirstName></FirstName>
               <LastName></LastName>
            </Contact>
         </ListOfContact>
      </Account>
   </ListOfAccount>
</SiebelMessage>
```

UpdateLeaves Method

Use UpdateLeaves to update existing records in the Siebel database. When UpdateLeaves is called on an integration object hierarchy, the EAI UI Data Adapter updates leaf nodes only and uses internal nodes for maintaining parent-child relationships.

Both internal nodes and leaf nodes must have Row Ids specified or the EAI UI Data Adapter generates an error. The EAI UI Data Adapter also generates an error if it does not find a match for the internal node and leaf node for a given Row Id.

UpdateLeaves Method Arguments

The following table lists the method arguments used with UpdateLeaves. For a complete description of the method arguments, see *EAI UI Data Adapter Business Service Method Arguments*.

Method Argument Name	Type
BusObjCacheSize	Input
LOVLanguageMode	Input
SiebelMessage	Input / Output

Example of Updating Root Component

The following example demonstrates updating a root component.

Request

The following is an example of a request:

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
   <ListOfAccount>
      <Account>1-1-1111</Account>
```

```
      <Employees>4900</Employees>
    </ListOfAccount>
</SiebelMessage>
```

Response

The following is an example of a response:

```
<SiebelMessage MessageId="P-3ITT" MessageType="Integration Object"
IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
   <ListOfAccount>
      <Account>
         <Id>1-1-1111</Id>
         <Mod_Id>2</Mod_Id>
      </Account>
   </ListOfAccount>
</SiebelMessage>
```

Example of Updating Child Component

This example demonstrates updating a child component.

Request

The following is an example of a request:

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
   <ListOfAccount>
      <Account>
         <Id>1-1-1111</Id>
         <Employees>5000</Employees>
         <ListOfBusiness_Address>
            <Business_Address>
               <Id>2-2-2222</Id>
               <Postal_Code>94404</Postal_Code>
            </Business_Address>
         </ListOfBusiness_Address>
      </Account>
   <ListOfAccount>
</SiebelMessage>
```

Response

The following is an example of a response:

```
<SiebelMessage MessageId="P-3ITW" MessageType="Integration Object"
IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
   <ListOfAccount>
      <Account>
         <Id>1-1-1111</Id>
         <Mod_Id>2</Mod_Id>
         <ListOfAccount_Business_Address>
            <Business_Address>
               <Id>2-2-2222</Id>
               <Mod_Id>2</Mod_Id>
            </Business_Address>
         </ListOfAccount_Business_Address>
      </Account>
   </ListOfAccount>
</SiebelMessage>
```

InitLeaves Method

Use InitLeaves to retrieve pre-default values. When InitLeaves is called on an integration object hierarchy, it retrieves the pre-default values for all leaf nodes. All internal nodes must exist in the database and Row Id must be specified.

InitLeaves Method Arguments

The following table lists the method arguments used with the InitLeaves Method. For a complete description of the method arguments, see *EAI UI Data Adapter Business Service Method Arguments*.

Method Argument	Type
BusObjCacheSize	Input
LOVLanguageMode	Input
SiebelMessage	Input / Output
ViewMode	Input

Example of Using InitLeaves on a Root Component

The following code snippet demonstrates using InitLeaves to retrieve default values for a root component. In this example the root component is Account.

Request

The following is an example of a request:

```
<?xml version="1.0" encoding="UTF-8"?>

<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
   <ListOfAccount>
      <Account>
         <Currency_Code></Currency_Code>
         <Account_Status></Account_Status>
         <Location_Type></Location_Type>
      </Account>
   </ListOfAccount>
</SiebelMessage>
```

Response

The following is an example of a response:

```
<?xml version="1.0" encoding="UTF-8"?>
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
   <ListOfAccount>
      <Account>
         <Account_Status>Active</Account_Status>
         <Currency_Code>USD</Currency_Code>
```

```
          <Location_Type>Corporate Training Center</Location_Type>
        </Account>
      </ListOfAccount>
    </SiebelMessage>
```

Example of Using InitLeaves on a Child Component

The following code snippets demonstrate using InitLeaves to retrieve pre-default values for a child component. In this example the parent component is Account and the child component is Business Address.

Request

The following is an example of a request:

```
<?xml version="1.0" encoding="UTF-8"?>
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
   <ListOfAccount>
      <Account>
         <Id>1-111112</Id>
         <ListOfBusiness_Address>
            <Business_Address>
               <Active_Status></Active_Status>
               <Main_Address_Flag></Main_Address_Flag>
            </Business_Address>
         </ListOfBusiness_Address>
      </Account>
   </ListOfAccount>
</SiebelMessage>
```

Response

The following is an example of a response:

```
<?xml version="1.0" encoding="UTF-8"?>
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
   <ListOfAccount>
      <Account>
         <ListOfBusiness_Address>
            <Business_Address>
               <Active_Status>Y</Active_Status>
               <Main_Address_Flag>Y</Main_Address_Flag>
            </Business_Address>
         </ListOfBusiness_Address>
      </Account>
   </ListOfAccount>
</SiebelMessage>
```

InsertLeaves Method

Use InsertLeaves to insert records into the Siebel database. When InsertLeaves is called on an integration object hierarchy, the EAI UI Data Adapter inserts leaf nodes only and uses internal nodes for maintaining parent-child relationships:

- **Internal Nodes.** All internal nodes must already exist in the database and Row Id must be specified (Row Id is the implicit, hard-coded user key used by the EAI UI Data Adapter). If the internal node does not exist or Row Id is not specified, then the EAI UI Data Adapter returns an error. For more information about user keys, see *About the EAI UI Data Adapter Business Service*.

- **Leaf Nodes.** Whether or not Row Id must be specified for leaf nodes depends on the type of integration component:
  - If the integration component represents a normal business component or MVG business component, Row Id must not be defined, because records for these components are being inserted.
  - If the integration component represents an association business component or an MVG association business component, leaf nodes might or might not have Row Ids defined. If Row Ids are specified, then the EAI UI Data Adapter creates an association record only. If Row Ids are not specified, then both a child record and an association record are created.

InsertLeaves returns an integration object hierarchy. Each integration component instance in the hierarchy has two fields: Row Id and Mod Id (implicit status keys used by the EAI UI Data Adapter). You can use these fields to retrieve the Row Id of the newly created record.

InsertLeaves Method Arguments

The following table lists the method arguments used with the InsertLeaves method. For descriptions of the methods, see *EAI UI Data Adapter Business Service Method Arguments*.

Method Argument Name	Type
BusObjCacheSize	Input
LOVLanguageMode	Input
SiebelMessage	Input / Output

Example of Inserting a Root Component

This example code snippet demonstrates inserting a non-existing account.

Request

The following is an example of a request:

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
   <ListOfAccount>
      <Account>
         <Type>Competitor</Type>
         <Name>Dixon Inc.</Name>
      </Account>
   </ListOfAccount>
</SiebelMessage>
```

Response

The following is an example of a response:

```
<SiebelMessage MessageId="P-3ITI" MessageType="Integration Object"
IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
   <ListOfAccount>
      <Account>
         <Id>P-5NA84</Id>
         <Mod_Id>0</Mod_Id>
      </Account>
```

```
    </ListOfAccount>
</SiebelMessage>
```

Example of Inserting a Child Component

The code snippets in this example demonstrate inserting a non-existing business address for an existing account.

Request

The following is an example of a request:

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Id>P-5NA84</Id>
      <ListOfBusiness_Address>
        <Business_Address>
          <City>San Carlos</City>
          <Street_Address>1145 laurel street</Street_Address>
          <State>CA</State>
          <Country>USA</Country>
          <Postal_Code>94063</Postal_Code>
        </Business_Address>
      </ListOfBusiness_Address>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```

Response

The following is an example of a response:

```
<SiebelMessage MessageId="P-3ITJ" MessageType="Integration Object"
IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Id>P-5NA84</Id>
      <Mod_Id>1</Mod_Id>
      <ListOfBusiness_Address>
        <Business_Address>
          <Id>P-5NA8B</Id>
          <Mod_Id>0</Mod_Id>
        </Business_Address>
      </ListOfBusiness_Address>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```

Example of Inserting an Association Child Component

This example demonstrate inserting an existing organization for an existing account. This operation associates the organization with the account. If the organization does not exist, then the EAI UI Data Adapter generates an error.

Request

The following is an example of a request:

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Id>P-5NA84</Id>
        <ListOfAccount_Organization>
```

```
            <Account_Organization>
               <Id>1-123</Id>
            </Account_Organization>
         </ListOfAccount_Organization>
      </Account>
   </ListOfAccount>
</SiebelMessage>
```

Response

The following is an example of a response:

```
<SiebelMessage MessageId="P-3ITL" MessageType="Integration Object"
IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
   <ListOfAccount>
      <Account>
         <Id>P-5NA84</Id>
         <Mod_Id>1</Mod_Id>
         <ListOfAccount_Organization>
            <Account_Organization IsPrimaryMVG="Y">
               <Id>0-R9NH</Id>
               <ModId>9</ModId>
            </Account_Organization>
            <Account_Organization IsPrimaryMVG="N">
               <Id>1-123</Id>
               <ModId>0</ModId>
            </Account_Organization>
         </ListOfAccount_Organization>
      </Account>
   </ListOfAccount>
</SiebelMessage>
```

DeleteLeaves Method

The DeleteLeaves method deletes leaf nodes only. If the Cascade Delete on the Link object is set to TRUE, then child records are also deleted. Row Ids are required for both internal nodes and leaf nodes. DeleteLeaves does not return a value when the operation is successful.

Method Arguments for DeleteLeaves

The following table lists the method arguments used with DeleteLeaves. For descriptions of the arguments, see *EAI UI Data Adapter Business Service Method Arguments*.

Method Argument Name	Type
IntObjectName	Input
LOVLanguageMode	Input
SiebelMessage	Input / Output

Example of Deleting a Root Component

This example demonstrates deleting a root component.

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
   <ListOfAccount>
      <Account>
         <Id>P-5NA84</Id>
      </Account>
   </ListOfAccount>
</SiebelMessage>
```

Example of Deleting a Child Component

This example demonstrates deleting a child component.

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
   <ListOfAccount>
      <Account>
         <Id>P-5NA84</Id>
         <ListOfBusiness_Address>
            <Business_Address>
            <Id>P-5NA8B</Id>
            </Business_Address>
         </ListOfBusiness_Address>
      </Account>
   </ListOfAccount>
</SiebelMessage>
```

Execute Method

The Execute method allows you to perform multiple operations on multiple business components. It is the only method that operates on internal nodes. The Execute method returns the same kind of object that the InsertLeaves method returns. For more information, see *InsertLeaves Method*.

> **Note:** the Execute method requires a status object only when it contains an insert operation on a child integration component instance. However, because the EAI UI Data Adapter processes in a top-down fashion, it adds a status object to the integration object instance even if an insert operation is not present.

The operations are defined by the *operation* attribute on the integration component instance. An integration component instance can have the following operations as defined in the following table.

Operation	Description
update	Updates the integration component instance
insert	Inserts the integration component instance
delete	Deletes the integration component instance
skipnode	Matches integration component instances and process children

CAUTION: Operations must be specified on every integration component instance. If an operation is not specified, then an implicit Synchronize operation will be performed, which will delete all unmatched child integration component instances.

Execute Method Arguments

The following table lists the method arguments used with the Execute method. For a description of the methods, see *EAI UI Data Adapter Business Service Method Arguments*.

Method Argument Name	Type
BusObjCacheSize	Input
LOVLanguageMode	Input
SiebelMessage	Input / Output

Example of Using the Execute Method

The following example demonstrates using the Execute method to perform update, insert, and delete operations on child object. Note that the skipnode operation is defined on the parent object.

Request

The following is an example of a request:

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
   <ListOfAccount>
      <Account operation="skipnode">
         <Id>1-1-1111</Id>
         <ListOfBusiness_Address>
            <Business_Address operation="update">
               <Id>2-2-2222</Id>
               <Postal_Code>94402</Postal_Code> <!--Postal Code changed-->
            </Business_Address>
            <Business_Address operation="insert">
               <Postal_Code>94402</Postal_Code>
               <City>San Mateo</City>
               <Street_Address>2215 Bridgepointe Parkway</Street_Address>
               <State>CA</State>
               <Country>USA</Country>
            </Business_Address>
         </ListOfBusiness_Address>
         <ListOfContact>
            <Contact operation="delete">
               <Id>4-4-4444</Id>
            </Contact>
         </ListOfContact>
      </Account>
   </ListOfAccount>
</SiebelMessage>
```

Response

The following is an example of a response:

```
<SiebelMessage MessageId="42-21YQ" MessageType="Integration Object"
IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
   <ListOfAccount>
      <Account>
         <Id>1-1-1111</Id>
         <Mod_Id>3</Mod_Id>
         <ListOfBusiness_Address>
            <Business_Address>
               <Id>2-2-2222</Id>
               <Mod_Id>1</Mod_Id>
            </Business_Address>
            <Business_Address>
               <Id>42-53Q2W</Id>
               <Mod_Id>0</Mod_Id>
            </Business_Address>
         </ListOfBusiness_Address>
      </Account>
   </ListOfAccount>
</SiebelMessage>
```

EAI UI Data Adapter Business Service Method Arguments

The methods exposed in the EAI UI Data Adapter business service take arguments that you use to specify information that the adapter uses when processing the request. The following table summarizes these method arguments.

Argument	Display Name	Description
BusObjCacheSize	Business Object Cache Size	Maximum Number of Business Objects that can be cached at one time.
ExecutionMode	Execution Mode	Used to set the direction of a query on a business component. Valid values are ForwardOnly and Bidirectional. The default is Bidirectional. ForwardOnly is more efficient than Bidirectional, and is recommended in cases where you must process a large number of records in the forward direction only (such as for report generation). For operations that are likely to return more than 10000 records, use ForwardOnly to avoid errors. For more information on executing queries, see the topic on the ExecuteQuery business component method in *Siebel Object Interfaces Reference* .
LOVLanguageMode	LOV Language Mode	Indicates whether the EAI UI Data Adapter must translate the LOV value before sending it to the object manager. Valid values are LIC or LDC. If LIC is specified, then the EAI UI Data Adapter expects language independent values in the input message and translates them to language dependent values (based on the current language setting) before the request is sent to the object manager. If LDC is specified, then the EAI UI Data Adapter does not translate the value before sending it to the object manager.

Argument	Display Name	Description
NamedSearchSpec	Predefined Query	Name of a PDQ. The EAI UI Data Adapter sets the name of the PDQ on the business object instance. If NamedSearchSpec and QBE are specified, then NamedSearchSpec is used.
NewQuery	New Query	Default is False. Boolean indicating whether a new query will be executed. If set to True, then a new query is executed flushing the cache for that particular integration object.
NumOutputObjects	Number of Output Integration Objects	Number of Integration Objects output
OutputIntObjectName	Not applicable	The name of the integration object that will be sent in the output.
SiebelMessage	Siebel Message	Input or output integration object instance.
ViewMode	View Mode	Visibility algorithm used in addition to a search specification to determine which records will be retrieved. The ViewMode method argument is used to set the View Mode property for all business components corresponding to the integration object. Valid values are *Manager*, *Sales Rep*, *Personal*, *Organization*, *Sub-Organization*, *Group*, *Catalog*, and *All.*

8 Siebel Virtual Business Components

Siebel Virtual Business Components

This chapter describes the virtual business component (VBC), its uses, and restrictions. This chapter also describes how you can create a new VBC in Siebel Tools. The following topics are included:

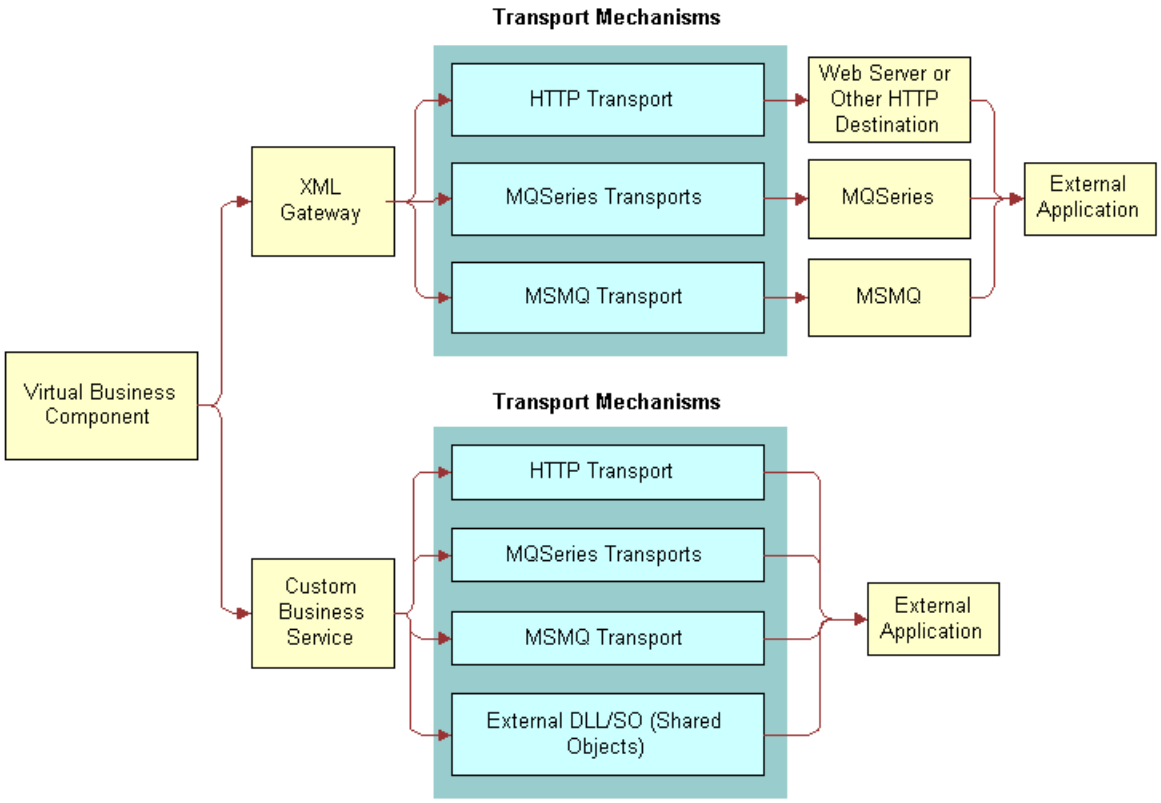
- *About Virtual Business Components*
- *Using Virtual Business Components*
- *XML Gateway Service*
- *Examples of the Outgoing XML Format*
- *Search-Spec Node-Type Values*
- *Examples of the Incoming XML Format*
- *External Application Setup*
- *Custom Business Service Methods*
- *Custom Business Service Examples*

About Virtual Business Components

A virtual business component (VBC) provides a way to access data that resides in an external data source using a Siebel business component. The VBC does not map to an underlying table in the Siebel Database. You create a new VBC in Siebel Tools workspace and deliver the workspace. The VBC calls a Siebel business service to provide a transport mechanism.

You can take two approaches to use VBCs, as illustrated in the following figure:

- Use the XML Gateway business service to pass data between the VBC and one of the Siebel transports, such as the EAI HTTP Transport or the EAI MSMQ Transport.
- Write your own business service in Siebel eScript or in Siebel VB to implement the methods described in this chapter.



Using VBCs for Your Business Requirements

The following features enhance the functionality of VBCs to better assist you in meeting your business requirements:

- VBCs support drilldown behavior:
  - You can drill down on a VBC to a standard business component, another VBC, or the same VBC.
  - You can drill down onto a VBC from a standard business component, another VBC, or the same VBC.
- A parent applet can be based on a VBC.
- You can define VBCs that can participate as a parent in a business object. The VBC you define can be a parent to a standard BC or a VBC.
- You still can use an older version of the XML format or property set by setting the VBC Compatibility Mode parameter to the appropriate version. For information, see the table in the topic *Setting User Properties for the Virtual Business Component* .
- You can pass search and sort specifications to the business service used by a VBC.
- You can use the Validation, Pre Default Value, Post Default Value, Link Specification, and No Copy attributes of the VBC fields.
- You can use predefined queries with VBCs.
- You can have picklists based on VBCs, and use picklist properties such as No Insert, No Delete, No Update, No Merge, Search Specification, and Sort Specification.
- You can use the Cascade Delete, Search Spec, Sort Spec, No Insert, No Update, and No Delete link properties when a VBC is the child business component on the link.

- You can use the No Insert, No Update, No Delete, Search Spec, Sort Spec, and Maximum Cursor Size business component properties.

Usage and Restrictions for Virtual Business Components

The following are the uses and restrictions of VBCs:

- You can define a business object as containing both standard business components and VBCs.
- When configuring applets based on VBCs, use CSSFrame (Form) and CSSFrameList (List) instead of specialized applet classes.
- (Optional) Using the same name for the VBC field names and the remote data source field names can reduce the amount of required programming.
- VBCs cannot be docked, so they do not apply to remote users.
- VBCs cannot contain a multivalue group (MVG).
- VBCs do not support many-to-many relationships.
- A pick applet based on a VBC instantiates the VBC without any parent reference (no link is used). As result, the VBC business service does not receive the source field value from the parent component If the VBC business service must access the current parent business component context, then you can use the ActiveBusObject method of the TheApplication object in a business service server script to do the following:
  - Obtain the current business object instance (assuming this is the instance with this VBC).
  - Instantiate the parent business component (assuming the name of the parent BC is known).
  - Obtain the parent business component field for referencing it as a source (the field must be active in the current parent business component).
- VBCs cannot be loaded using Enterprise Integration Manager.
- Standard business components cannot contain multivalue groups based on VBCs.
- VBCs cannot be implemented using any business component class other than CSSBCVExtern. This means specialized business components such as Quotes and Forecasts cannot be implemented as VBCs.
- You cannot use Workflow Monitor to monitor VBCs.
- You cannot execute queries against VBCs when the search specification uses a function that is normally supported for Query mode against regular business components, such as ParentFieldValue().

Using Virtual Business Components

To use VBCs to share data with an external application, perform the following high-level tasks:

- *Creating a New Virtual Business Component*
- *Setting User Properties for the Virtual Business Component*
- Configuring your VBC business service:
  - Configure your XML Gateway Service or write your own business service.
    For information, see *XML Gateway Service* and *Custom Business Service Methods*.
  - Configure your external application.

For information, see *External Application Setup*.

Creating a New Virtual Business Component

You create a new VBC in Siebel Tools.

To create a new virtual business component

1. In Siebel Tools, lock the appropriate project.
2. In the Object Explorer, select the Business Component object.
3. Right-click, and then choose New Record.
4. Name the business component.
5. Select the project you locked in Step 1.
6. Set the class to CSSBCVExtern. This class provides the VBC functionality.

Setting User Properties for the Virtual Business Component

When defining the VBC, you must provide the user properties shown in the following table.

User Property	Description
Service Name	The name of the business service.
Service Parameters	(Optional) Any parameters required by the business service. The Siebel application passes this user property, as an input argument, to the business service.
Remote Source	(Optional) External data source that the business service is to use. This property allows the VBC to pass a root property argument to the underlying business service, but it does not allow a connection directly to the external datasource. The Siebel application passes only this user property as an input argument.
VBC Compatibility Mode	(Optional) Determines the format of the property set passed from a VBC to a business service, or the format in which the outgoing XML from the XML Gateway will be. A valid value is Siebel *xxx*, where *xxx* can be any Siebel release number. Some examples would be Siebel 6 or Siebel 7.0.4. If *xxx* is less than 7.5, the format will be in a release that is earlier than release 7.5. Otherwise, a new property set, and the XML format will be passed. If you are creating a VBC in version 7.5 or higher, then it is not necessary to define this new user property, because the default is to use the new PropertySet from a VBC and the new outgoing XML from the XML Gateway. For your existing VBC implementation, update your VBC definition by adding this new user property, and setting it to Siebel *xxx*, where *xxx* is the version number that you want.

To define user properties for a virtual business component

1. In the Object List Editor in Siebel Tools, select the virtual business component for which you want to define user properties.
2. In the Object Explorer, expand the Business Component tree, and then select Business Component User Prop.

3. In the Object List Editor, click in the Business Component User Props list, right-click, and then choose New Record.
4. Type the name of the user property, such as Service Name, in the Name field.
5. Type the value of the user property, such as a business service name, in the Value field.
6. Repeat the process for every user property you want to define for this VBC.

Note: For the list of different property sets and their format, see *Examples of the Outgoing XML Format* and *Examples of the Incoming XML Format*.

XML Gateway Service

The XML Gateway business service communicates between Siebel CRM and external data sources using XML as the data format. For information on XML format, see *Examples of the Outgoing XML Format* and *Examples of the Incoming XML Format*. The XML Gateway business service can be configured to use one of the following transports:

- EAI MQSeries Server Transport
- EAI HTTP Transport
- EAI MSMQ Transport

You can configure the XML Gateway by specifying the transport protocol and the transport parameters you use in the Service Parameters User Property of the VBC, as shown in the following table. When using the XML Gateway, specify the following user properties for your VBC.

Name	Value
Service Name	XML Gateway
Service Parameters	variable1 name=variable1 value; variable2 name=variable2 value>;...
Remote Source	*External Data Source*
VBC Compatibility Mode	Siebel *xxx*, where *xxx* can be any Siebel release number.

Note: You can concatenate multiple name-value pairs using a semicolon (;), but do not use any spaces between the name, the equal sign, the value, and the semicolon.

For example, if you want to specify the EAI HTTP Transport, then you can use something like the following:

```
"Transport=EAI HTTP Transport;HTTPRequestURLTemplate=<your
URL>;HTTPRequestMethod=POST"
```

You can also implement a VBC with MQSeries. The following procedure lists the steps you take to implement this.

To implement a VBC with MQSeries

1. Call the EAI Business Integration Manager (Server Request) business service.

2. Define another service parameter for the name of a workflow to run, with the following user properties on the VBC:
    - **Service Name.** XML Gateway.
    - **Service Parameters.** Transport=EAI Business Integration Manager (Server Request);ProcessName=EAITEST.
3. Define a workflow, EAITEST, to call the EAI MQSeries Server Transport with the SendReceive method.
4. Define a new process property, <Value>, on the workflow, and use it as an output argument on the EAI MQSeries Server Transport step in the workflow.

XML Gateway Methods

The XML Gateway provides the methods presented in the following table.

Method	Description
Init	Initializes the XML Gateway business service for every business component.
Delete	Deletes a given record in the remote data source.
Insert	Inserts a record into a remote data source.
PreInsert	Performs an operation that tests for the existence of the given business component. Only default values are returned from the external application.
Query	Queries the given business component from the given data source.
Update	Updates a record in the remote data source.

XML Gateway Method Arguments

The XML Gateway init, delete, insert, preInsert, query, and update methods take the arguments presented in the following table.

Argument	Description
Remote Source	The VBC Remote Source user property. The remote source from which the service is to retrieve data for the business component. This must be a valid connect string. When configuring the repository business component on top of the specialized business component class CSSBCVExten, you can define a user property Remote Source to allow the Transport Services to determine the remote destination and any connect information. If this user property is defined, then it is passed to every request as the remote-source tag.
Business Component Id	Unique key for the given business component.

Argument	Description
Business Component Name	Name of the business component or its equivalent, such as a table name.
Parameters	The VBC Service Parameters user property. A set of string parameters required for initializing the XML Gateway.

About Handling White Space

White space is handled by the XML parser while processing the request from the XML Gateway business service. When the white space is part of the XML syntax, it must be discarded by the XML parser and not *preserved* (passed on to the processing application). If the white space is in any other location (such as in element content within a document), then it must be preserved according to the XML specification, because it might have some meaning.

For example:

```
<mydata>
<mytag>stuff</mytag>
</mydata>
```

and

```
<mydata><mytag>stuff</mytag></mydata>
```

are different to an XML parser.

To preserve white space, use the `xml:space` attribute with the value `preserve`, for example:

```
<mydata xml:space="preserve">
<mytag>stuff and more stuff</mytag>
</mydata>
```

The value of `xml:space` applies to all children of the element containing the attribute unless overridden by one of the children.

For more information on white space handling and the `xml:space` attribute, see Microsoft Developer Network (http://msdn.microsoft.com).

Examples of the Outgoing XML Format

Examples of the XML documents generated and sent by the XML Gateway to the external system are presented in the following table. These examples are based on the Siebel eScript example in *Custom Business Service Examples*. See *DTDs for XML Gateway Business Service* for examples of the DTDs that correspond to each of these methods.

> **Note:** The XML examples in this chapter have extraneous carriage returns and line feeds for ease of reading. Delete all the carriage returns and line feeds before using any of the examples.

Method	Format of the XML Stream	Description
Delete Request	<siebel-xmlext-delete-req> <buscomp id="1">Contact</buscomp> <remote-source>http://throth/servlet/VBCContacts</remote-source> <row> <value field="AccountId">146</value> <value field="Name">Max Adams</value> <value field="Phone">(408)234-1029</value> <value field="Location">San Jose</value> <value field="AccessId">146</value> </row> </siebel-xmlext-delete-req>	siebel-xmlext-delete-req This tag requests removal of a single record in the remote system.
Init Request	<siebel-xmlext-fields-req> <buscomp id="1">Contact</buscomp> <remote-source>http://throth/servlet/VBCContacts</remote-source> </siebel-xmlext-fields-req>	• siebel-xmlext-fields-req This tag fetches the list of fields supported by this instance. • buscomp Id The business component ID. • remote-source The remote source from which the service is to retrieve data for the business component.
Insert Request	<siebel-xmlext-insert-req> <buscomp id="1">Contact</buscomp> <remote-source>http://throth/servlet/VBCContacts</remote-source> <row> <value field="AccountId">1-6</value> <value field="Name">Max Adams</value> <value field="Phone">(398)765-1290</value> <value field="Location">Troy</value> <value field="AccessId"></value> </row> </siebel-xmlext-insert-req>	siebel-xmlext-insert-req This tag requests the commit of a new record in the remote system. The insert-req XML stream contains values for fields entered through the business component.

Method	Format of the XML Stream	Description

PreInsert Request	`<siebel-xmlext-preinsert-req>`
	
	`<buscomp id="1">Contact</buscomp>`
	
	`<remote-source>http://throth/servlet/VBCContacts</remote-source>`
	
	`</siebel-xmlext-preinsert-req>`
		siebel-xmlext-preinsert-req
		
		This tag allows the connector to provide default values. This operation is called when a new row is created, but before any values are entered through the business component interface.

Query Request	`<siebel-xmlext-query-req>`
	
	`<buscomp id="1">Contact</buscomp>`
	
	`<remote-source>http://throth/servlet/VBCContacts</remote-source>`
	
	`<max-rows>6/max-rows>`
	
	`<search-string>=([Phone] IS NOT NULL) AND ([AccountId] = "1-6")</search-string>`
	
	`<search-spec>`
	
	`<node node-type="Binary Operator">AND`
	
	`<node node-type="Unary Operator">IS NOT NULL`
	
	`<node node- type="Identifier">Phone</node></node>`
	
	`<node node-type="Binary Operator">=`
	
	`<node node-type="Identifier">AccountId</node>`
	
	`<node value-type="TEXT" node-type="Constant">1-6</node>`
	
	`</node>`
	
	`</node>`
	
	`</search-spec>`
	
	`<sort-spec>`
	
	`<sort field="Location">ASCENDING</sort>`
	
	`<sort field="Name">DESCENDING</sort>`
	
	`</sort-spec>`
	
	`</Siebel-xmlext-query-req>`
		• siebel-xmlext-query-req
		This tag queries by example. The query-req XML stream contains parameters necessary to set up the query. In this example, the query requests that record information be returned from the remote system.
		
		• max-rows
		Maximum number of rows to be returned. The value is the Maximum Cursor Size defined at the VBC plus one. If the Maximum Cursor Size property is not defined at the VBC, then the max-rows property is not passed.
		
		• search-string
		The search specification used to query and filter the information.
		
		• search-spec
		Hierarchical representation of the search-string. For information, see *Search-Spec Node-Type Values*.
		
		• sort-spec
		List of sort fields and sort order.
		
		Note: In some cases you might retrieve external data for display in a child list applet, using a link to a parent business component. If the parent business component field on which the link is based is empty, then the query request is sent without a search-spec tag, but instead with the following tag: `match field="` *Child BC Fieldname* `"/>`

Update Request	`<buscomp id="2">Contact</buscomp>`	siebel-xmlext-update-req

Method	Format of the XML Stream	Description
	<remote-source>http://throth/servlet/VBCContacts</remote-source> <row> <value changed="false" field="AccountId">1-6</value> <value changed="false" field="Name">Max Adams</value> <value changed="true" field="Phone">(408)234-1029</value> <value changed="true" field="Location">San Jose</value> <value changed="false" field="AccessId">146</value> </row> </siebel-xmlext-update-req>	This tag requests changes to the field values for an existing row. All values for the record are passed with the value tags, and with the changed attribute identifying the ones that have been changed through the Siebel application.

Search-Spec Node-Type Values

The search-string is in the Siebel query language format. The search-string is parsed by the Siebel query object and then turned into the hierarchical search-spec. The following table shows the different search-spec node-types and their values.

Node-Type	PropertySet and XML Representation
Constant	Example: <node node-type = "Constant" value-type="NUMBER">1000</node> The valid value-types are TEXT, NUMBER, DATETIME, UTCDATETIME, DATE, and TIME.
Identifier	Example: <node node-type="Identifier">Name</node> The value Name is a valid business component field name.
Unary Operator	Example: <node node-type="Unary Operator">NOT</node> The valid values are NOT, EXISTS, IS NULL, IS NOT NULL.
Binary Operator	Example: <node node-type= "Binary Operator" >AND</node> The valid values are LIKE, NOT LIKE, SOUNDSLIKE, =, <>, <=, <, >=, >, AND, OR, +, -, *, /, ^.

Examples of the Incoming XML Format

The following table contains examples of XML documents that are sent from an external system to the XML Gateway in response to a request. These examples are based on the Siebel eScript example in *Custom Business Service Examples*. See *DTDs for XML Gateway Business Service* for examples of the DTDs that correspond to each of these methods.

Method	Format of the XML Stream	Description
Delete Return	`<siebel-xmlext-delete-ret/>`	siebel-xmlext-delete-ret. Only the XML stream tag is returned.

Error	`<siebel-xmlext-status>`
	
	`<status-code>4</code>`
	
	`<error-field>Name</error-field>`
	
	`<error-text>Name must not be empty</ error-text>`
	
	`</siebel-xmlext-status>`
		Format of the XML stream expected by the Siebel application in case of an error in the external application. If the error is specific to a field, then the field name must be specified.
		
		The tags for this XML stream, and the entire XML stream, are optional:
		
		• siebel-xmlext-status
		
		This tag is used to check the status returned by the external system.
		
		• status-code
		
		This tag overrides the return value.
		
		• error-text
		
		This tag specifies textual representation of the error, if it is available. This tag appears in addition to the standard error message. For example, if the Siebel application attempts to update a record in the external system with a NULL Name, and this is not allowed in the external system, then the error text is set to: "Name must not be empty."

Init Return	`<siebel-xmlext-fields-ret>`
	
	`<support field="AccountId"/>`
	
	`<support field="Name"/>`
	
	`<support field="Phone"/>`
	
	`<support field="Location"/>`
	
	`<support field="AccessId"/>`
	
	`</siebel-xmlext-fields-ret>`
		siebel-xmlext-fields-ret
		
		The fields-ret XML stream return contains the list of VBC fields supported by the external application for this instance.
		
		The following field names are reserved by the Siebel application, and must not appear in this list:
		
		• Id
		• Created
		• Created By
		• Updated
		• Updated By

Method	Format of the XML Stream	Description
Insert Return	<siebel-xmlext-insert-ret> <row> <value field="AccountId">1-6</value> <value field="Name">Max Adams</value> <value field="Phone">(398)765-1290</value> <value field="Location">Troy</value> <value field="AccessId">146</value> </row> </siebel-xmlext-insert-ret>	siebel-xmlext-insert-ret If the remote system has inserted records, then they can be returned to be reflected in the business component in an insert-ret XML stream in the row tag format as the insert-ret stream.
PreInsert Return	<siebel-xmlext-preinsert-ret> <row> <value field="Location">San Jose</value> </row> </siebel-xmlext-preinsert-ret>	siebel-xmlext-preinsert-ret Returns default values for each field, if there is any default value.
Query Return	<siebel-xmlext-query-ret> <row> <value field="AccountId">1-6</value> <value field="Name">Sara Chen</value> <value field="Phone">(415)298-7890</value> <value field="Location">San Francisco</value> <value field="AccessId">128</value> </row> <row> <value field="AccountId">1-6</ value> <value field="Name">Eric Brown</value> <value field="Phone">(650)123-1000</value> <value field="Location">Palo Alto </value> <value field="AccessId">129</value>	• siebel-xmlext-query-ret The query-ret XML stream contains the result set that matches the criteria of the query. • row This tag indicates the number of rows returned by the query. Each row must contain one or more value tags. The attributes that appear in row tags must be able to uniquely identify the rows. If there is a unique key in the remote data source, then it appears in the result set. If not, a unique key is generated. It is necessary to identify specific rows for DML operations. • value This tag specifies the field and value pairs and must be the same for each row in the set.

Method	Format of the XML Stream	Description

	`</row>`
	
	`</siebel-xmlext-query-ret>`	

Update Return	`<siebel-xmlext-update-ret>`
	
	`<row>`
	
	`<value field="Location">San Jose</value>`
	
	`<value field="Phone">(408)234-1029</value>`
	
	`</row>`
	
	`</siebel-xmlext-update-ret>`
		siebel-xmlext-update-ret
		
		If the remote system updated fields, then the fields can be returned to be reflected in the business component in an update-ret XML stream in the row tag format as the update-ret stream.

External Application Setup

When you have your XML Gateway Service configured, set up your external application accordingly to receive and respond to the requests. At a minimum, the external application must support the Init() and Query() methods, and depending upon the functionality provided by the VBC, the remaining methods might or might not be necessary.

Custom Business Service Methods

Your business service must implement the Init and Query methods as described in this topic. The Delete, PreInsert, Insert, and Update methods are optional, and depend on the functionality required by the VBC.

Note: Custom business services can be based only on the CSSService class, as specified in Siebel Tools.

These methods pass property sets between the VBC and the business service. VBC methods take property sets as arguments. Each method takes two property sets: an Inputs property set and an Outputs property set. The methods are called by the *CSSBCVExtern* class in response to requests from other objects that refer to, or are based on the VBC.

If VBCs are used, then custom business services are written to access external relational databases. However, it is recommended that you use external business components (EBCs) to access these databases instead of writing custom business services. For more information on EBCs, see *External Business Components*.

Common Method Parameters

The following table shows the input parameters common to every method. Note that all these parameters are at the root property set.

Parameter	Description
Remote Source	(Optional) Specifies the name of an external data source. This is the VBC's Remote Source user property, if defined. For information, see the table in topic *Setting User Properties for the Virtual Business Component*.
Business Component Name	Name of the active VBC.
Business Component Id	Internally generated unique value that represents the VBC.
Parameters	(Optional) The VBC's Service Parameters user property, if defined. For information, see the table in topic *Setting User Properties for the Virtual Business Component*. A set of parameters required by the business service.
VBC Compatibility Mode	(Optional) This is the VBC's Compatibility Mode user property, if defined. For information, see the table in topic *Setting User Properties for the Virtual Business Component*.

When a response has been received, the method packages the response from the external data source into the output's property set.

Business Services Methods and Their Property Sets

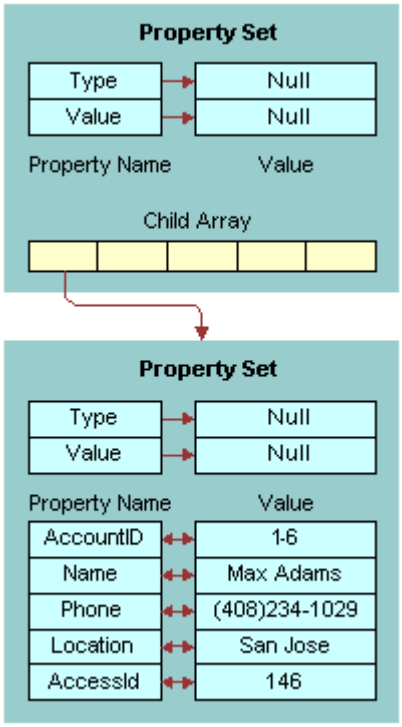
The following examples display each method's input and output property sets for a VBC Contact that displays simple contact information for a given account. These examples are based on the example in the *Custom Business Service Examples*.

The output property set of the Insert and Update methods for VBC does not affect the data in the business component, unlike the Query method, which uses the output property set to populate the business component. The output property set for Insert and Update is used to indicate that what fields or record has been changed.

Note: All the optional parameters have been omitted from these examples to simplify them.

Delete Method

The Delete method is called when a record is deleted. The following figure illustrates the property set for the Delete input.

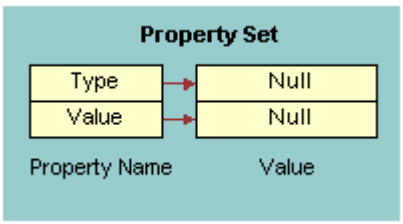


The following is the XML representation of the property set shown in the previous figure:

```
<?xml version="1.0" encoding="UTF-8" ?>

    <?Siebel-Property-Set EscapeNames="true"?>
    <PropertySet
     Business_spcComponent_spcId="1"
     Business_spcComponent_spcName="Contact">
     <PropertySet
     AccountId="1-6"
     Name="Max Adams"
     Phone="(408)234-1029"
     Location="San Jose"
     AccessId="146" />
     </PropertySet>
```

The following figure illustrates the property set for the Delete output.

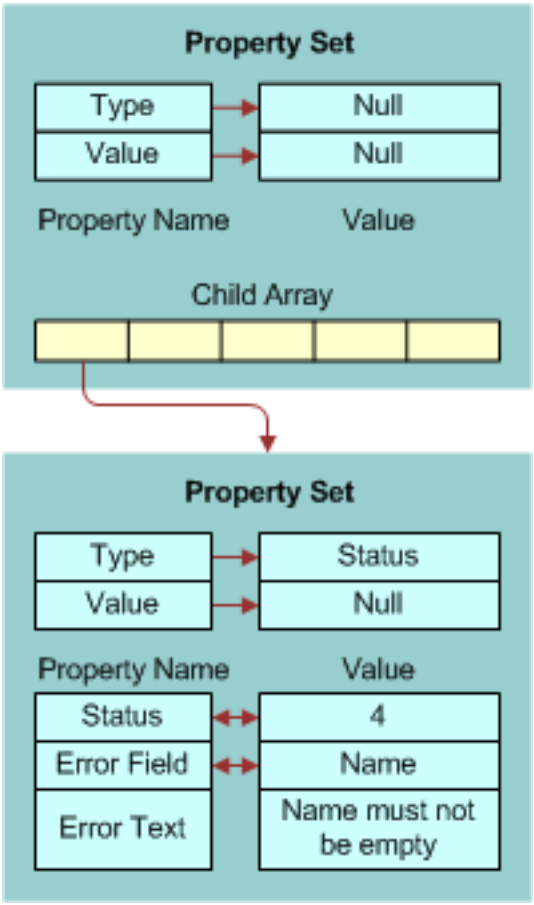


The following is the XML representation of the property set shown in the previous figure:

```
    <?xml version="1.0" encoding="UTF-8" ?>
    <?Siebel-Property-Set EscapeNames="true"?>
    <PropertySet />
```

Error Return Method

The following figure illustrates the property set for the Error Return, when an error is detected.



The following is the XML representation of the property set shown in the previous figure:

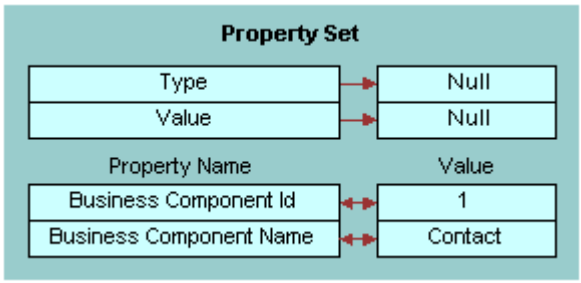
```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet>
 <Status Status="4"
 Error_spcField="Name"
 Error_spcText="Name must not be empty"/>
</PropertySet>
```

Init Method

The Init method is called when the VBC is first instantiated. It initializes the VBC. It expects to receive the list of fields supported by the external system.

Note: When a field is not initialized in the Init method of the VBC, the Update method is not fired when the field gets updated.

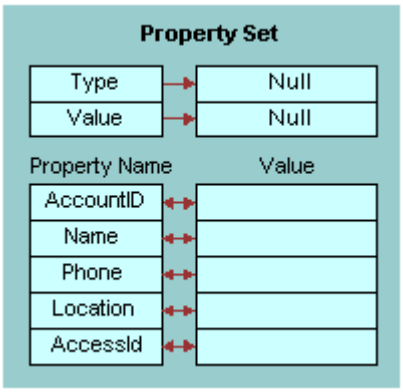
The following figure illustrates the property set for the Init input.



The following is the XML representation of the property set shown in the previous figure:

```
<?xml version="1.0" encoding="UTF-8"?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet
 Business_spcComponent_spcId="1"
 Business_spcComponent_spcName="Contact"/>
```

The following figure illustrates the property set for the Init output.

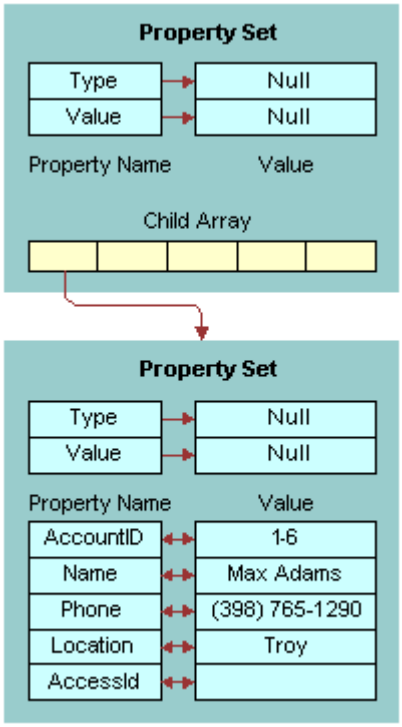


The following is the XML representation of the property set shown in the previous figure:

```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet
 AccountId=""
 Name=""
 Phone=""
 Location=""
 AccessId="" />
```

Insert Method

The Insert method is called when a New Record is committed. The following figure illustrates the property set for the Insert input.

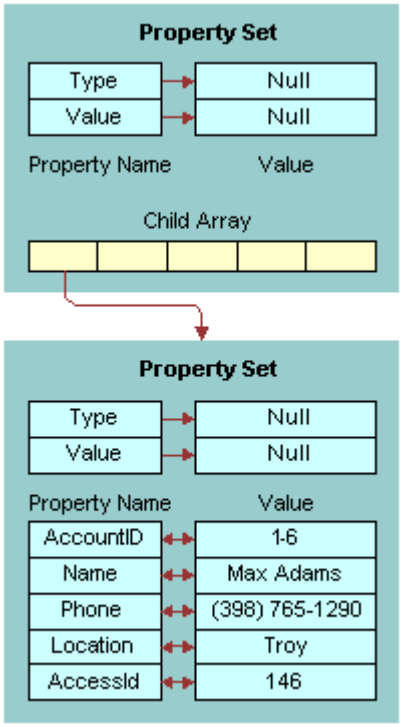


The following is the XML representation of the property set shown in the previous figure:

```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet
 Business_spcComponent_spcId="1"
 Business_spcComponent_spcName="Contact">
 <PropertySet
 AccountId="1-6"
 Name="Max Adams"
 Phone="(398)765-1290"
 Location="Troy"
 AccessId="" />
</PropertySet>
```

The following figure illustrates the property set for the Insert output.

Note: The property set for the Insert output does not affect the data in the business component. The output property set for Insert is used to indicate what fields or records were changed.

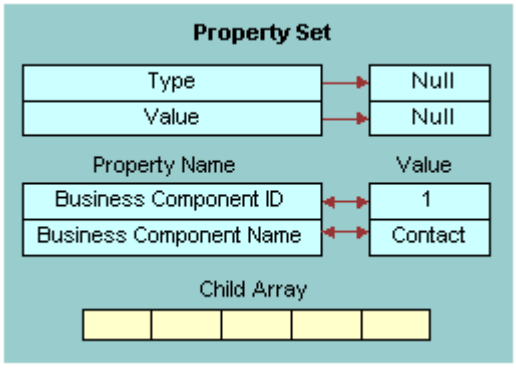


The following is the XML representation of the property set shown in the previous figure:

```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet
 <PropertySet
 AccountId="1-6"
 Name="Max Adams"
 Phone="(398)765-1290"
 Location="Troy"
 AccessId="146" />
</PropertySet>
```

PreInsert Method

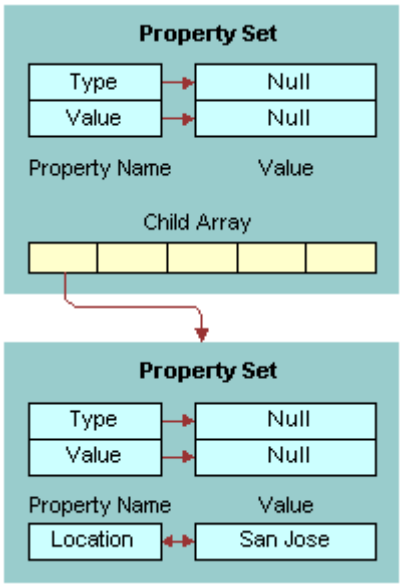
The PreInsert method is called when a New Record operation is performed. It supplies default values. The following figure illustrates the property set for the PreInsert input.



The following is the XML representation of the property set shown in the previous figure:

```
<?xml version="1.0" encoding="UTF-8"?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet
 Business_spcComponent_spcId="1"
 Business_spcComponent_spcName="Contact"/>
```

The following figure illustrates the property set for the PreInsert output.

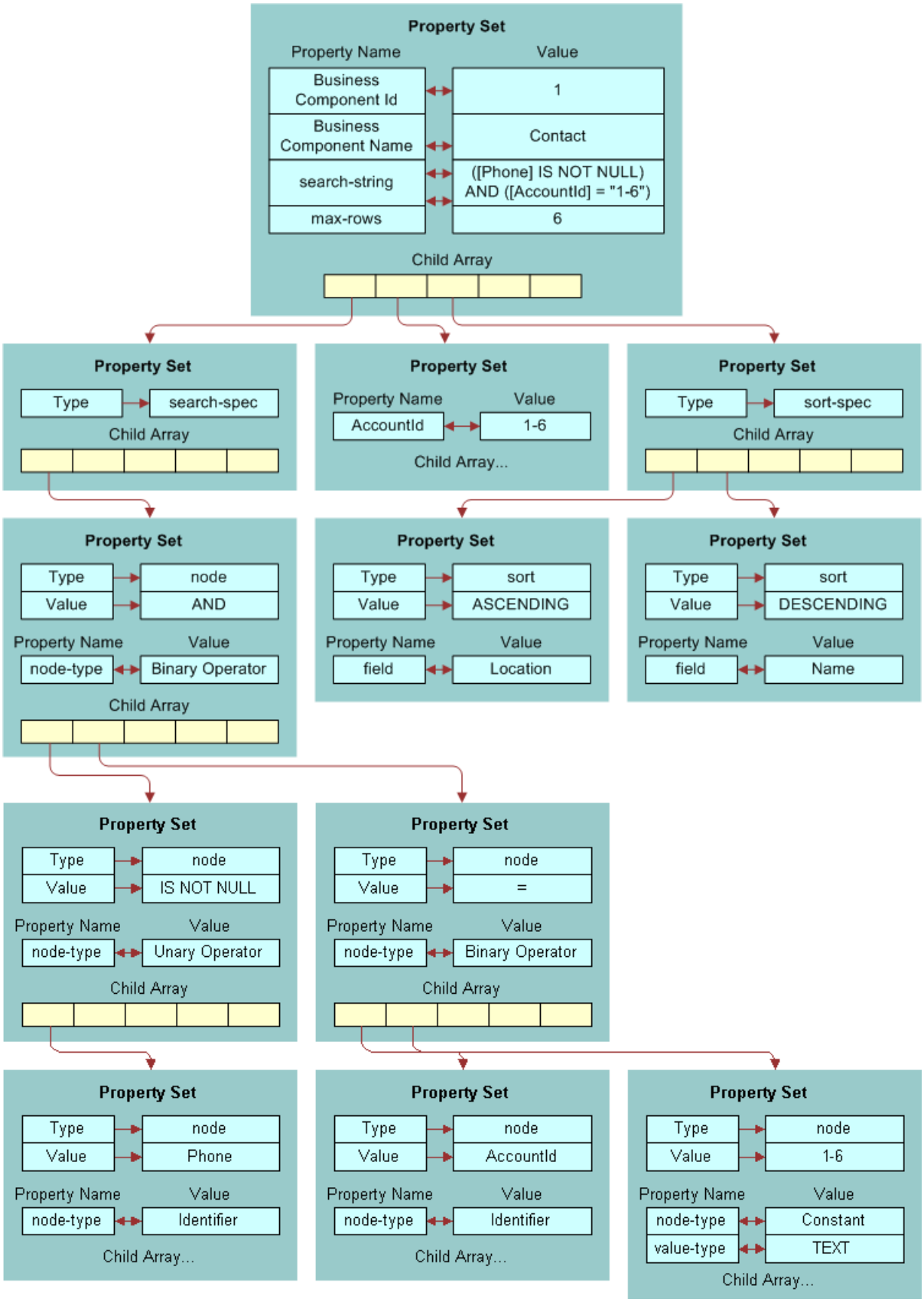


The following is the XML representation of the property set shown in the previous figure:

```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet>
 <PropertySet Location="San Jose" />
</PropertySet>
```

Query Method

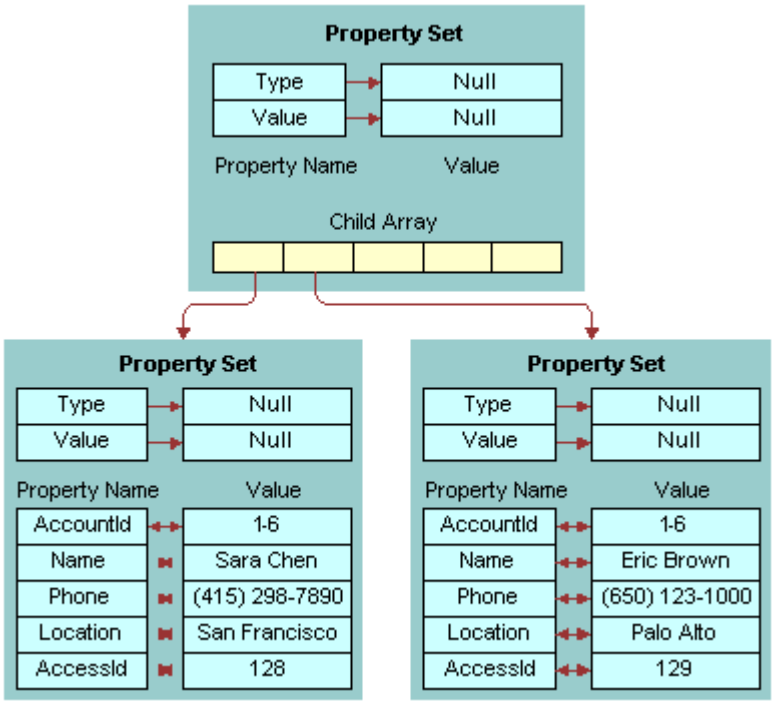
The Query method is called when a search is performed. The Query method must be supported by every VBC. Each record that matches the query is represented as a property set. For example, if 5 records match the query, then there will be 5 child property sets. Each property set contains a list of field names, that is field value pairs representing the values of each field for that particular record. The following figure illustrates the property set for the Query input and is followed by its XML representation.


Property Set
Property Name
Value
Business Component Id
1
Business Component Name
Contact
search-string
([Phone] IS NOT NULL) AND ([AccountId] = "1-6")
max-rows
6
Child Array
Property Set
Type
search-spec
Child Array
Property Set
Property Name
Value
AccountId
1-6
Child Array...
Property Set
Type
sort-spec
Child Array
Property Set
Type
node
Value
AND
Property Name
Value
node-type
Binary Operator
Child Array
Property Set
Type
sort
Value
ASCENDING
Property Name
Value
field
Location
Property Set
Type
sort
Value
DESCENDING
Property Name
Value
field
Name
Property Set
Type
node
Value
IS NOT NULL
Property Name
Value
node-type
Unary Operator
Child Array
Property Set
Type
node
Value
=
Property Name
Value
node-type
Binary Operator
Child Array
Property Set
Type
node
Value
Phone
Property Name
Value
node-type
Identifier
Child Array...
Property Set
Type
node
Value
AccountId
Property Name
Value
node-type
Identifier
Child Array...
Property Set
Type
node
Value
1-6
Property Name
Value
node-type
Constant
value-type
TEXT
Child Array...

The following is the XML representation of the property set shown in the previous figure:

```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet
 max-rows="6"
 search-string="([Phone] IS NOT NULL) AND ([AccountId] = "1-6")"
 Business_spcComponent_spcId="1"
 Business_spcComponent_spcName="Contact">
 <PropertySet AccountId="1-6" />
 <search-spec>
 <node node-type="Binary Operator">AND
 <node node-type="Unary Operator">IS NOT NULL
 <node node-type="Identifier">Phone</node>
 </node>
 <node node-type="Binary Operator">=
 <node node-type="Identifier">AccountId</node>
 <node value-type="TEXT" node-type="Constant">1-6</node>
 </node>
 </node>
</search-spec>
<sort-spec>
 <sort field="Location">ASCENDING</sort>
 <sort field="Name">DESCENDING</sort>
 </sort-spec>
</PropertySet>
```

The following figure illustrates the property set for the Query output.



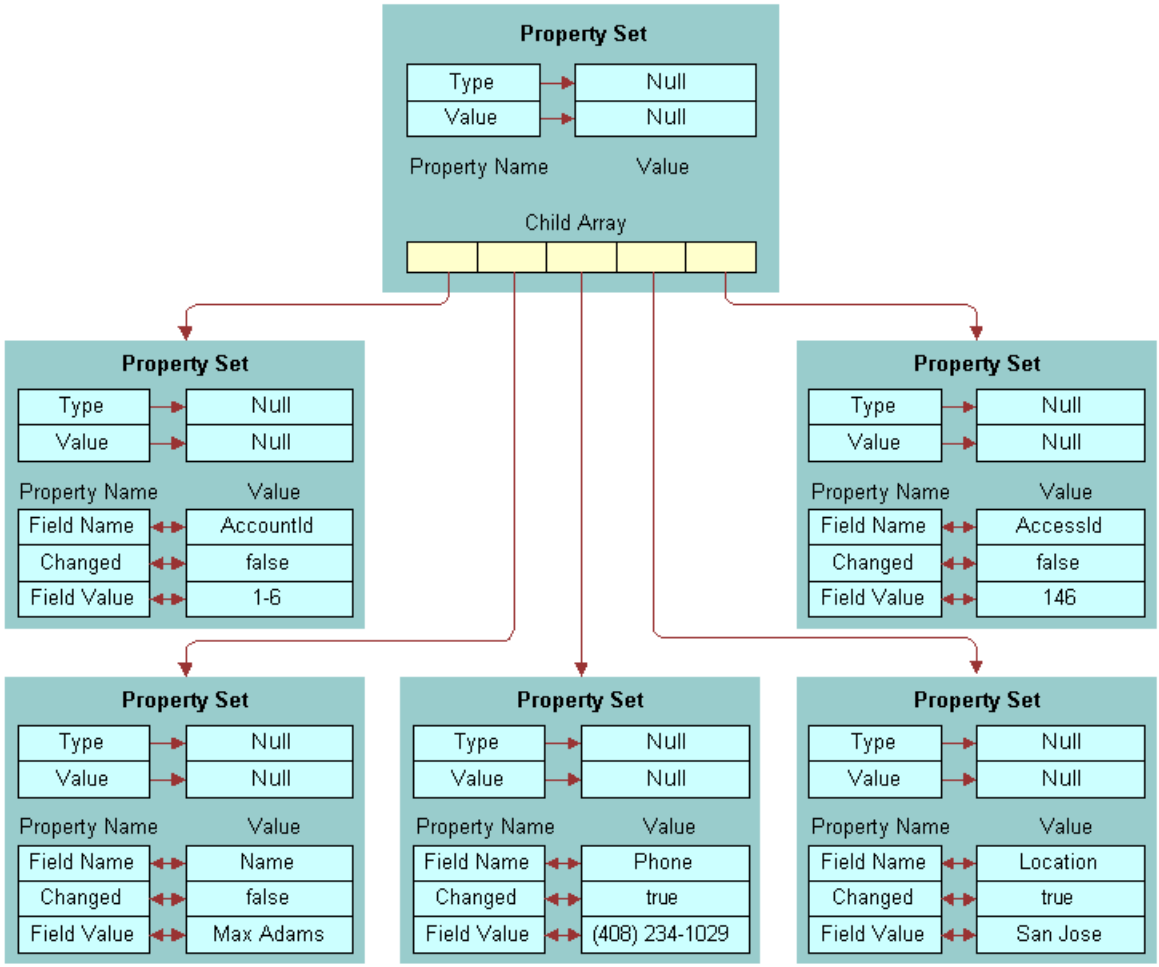
The following is the XML representation of the property set shown in the previous figure:

```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet>
 <PropertySet
 AccountId="1-6"
 Name="Sara Chen"
 Phone="(415)298-7890"
```

```
 Location="San Francisco"
 AccessId="128" />
<PropertySet
 AccountId="1-6"
 Name="Eric Brown"
 Phone="(650)123-1000"
 Location="Palo Alto"
 AccessId="129" />
</PropertySet>
```

Update Method

The Update method is called when a record is modified. The following figure illustrates the property set for the Update input.



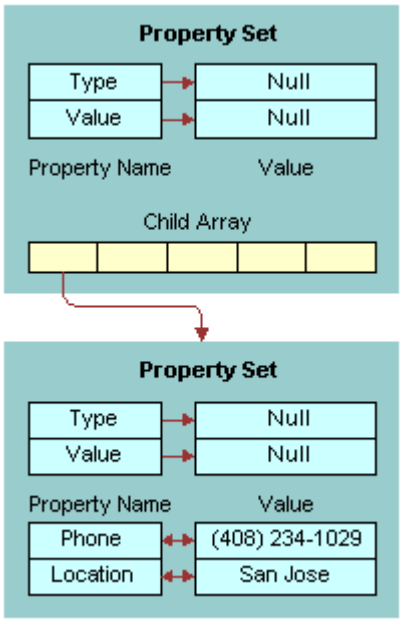
The following is the XML representation of the property set shown in the previous figure:

```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet
 Business_spcComponent_spcId="1"
 Business_spcComponent_spcName="Contact">
<PropertySet
 Field_spcName="AccountId"
```

```
    Changed="false"
    Field_spcValue="1-6"/>
   <PropertySet
    Field_spcName="Name"
    Changed="false"
    Field_spcValue="MaxAdams"/>
   <PropertySet
    Field_spcName="Phone"
    Changed="true"
    Field_spcValue="(408)234-1029"/>
   <PropertySet
    Field_spcName="Location"
    Changed="true"
    Field_spcValue="SanJose"/>
   <PropertySet
    Field_spcName="AccessId"
    Changed="false"
    Field_spcValue="146" />
   </PropertySet>
```

The following figure illustrates the property set for the Update output.

Note: The property set for Update output does not affect the data in the business component. The output property set for Update is used to indicate what fields or records were changed.



The following is the XML representation of the property set shown in the previous figure:

```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet
 <PropertySet
 Phone=="(408)234-1029"
 Location="San Jose" />
</PropertySet>
```

Custom Business Service Examples

These examples show the implementation of a business service for a VBC in both Siebel eScript and Siebel VB:

- *Siebel eScript Business Service Example for a VBC*
- *Siebel VB Business Service Example for a VBC*

Siebel eScript Business Service Example for a VBC

The following is an example of Siebel eScript implementation of a business service for a VBC. The fields configured for this simple VBC are AccountId, Name, Phone, Location, and AccessId. AccessId is the primary key in the external data source. AccessId is included in the VBC fields to make updating and deleting the fields simple and is configured as a hidden field.

> **CAUTION:** Do not use Siebel CRM system fields, such as Id, as output properties. Undesired application behavior might result.

```
function Service_PreInvokeMethod (MethodName, Inputs, Outputs) {
if (MethodName == "Init") {
return(Init(Inputs, Outputs));
}
else if (MethodName == "Query") {
return(Query(Inputs, Outputs));
}
else if (MethodName == "PreInsert") {
return(PreInsert(Inputs, Outputs));
}
else if (MethodName == "Insert") {
return(Insert(Inputs, Outputs));
}
else if (MethodName == "Update") {
return(Update(Inputs, Outputs));
}
else if (MethodName == "Delete") {
return(Delete(Inputs, Outputs));
}
else {
return (ContinueOperation);
}
}
function Init (Inputs, Outputs) {
// For debugging purposes...
logPropSet(Inputs, "InitInputs.xml");
Outputs.SetProperty("AccountId", "");
Outputs.SetProperty("Name", "");
Outputs.SetProperty("Phone", "");
Outputs.SetProperty("AccessId", "");
Outputs.SetProperty("Location", "");
// For debugging purposes...
logPropSet(Outputs, "InitOutputs.xml");
return (CancelOperation);
}
function Query(Inputs, Outputs) {
// For debugging purposes...
logPropSet(Inputs, "QueryInputs.xml");
var selectStmt = "select * from Contacts ";
```

```
var whereClause = "";
var orderbyClause = "";
// You have the following properties if you want to use them
// Inputs.GetProperty("Business Component Name")
// Inputs.GetProperty("Business Component Id")
// Inputs.GetProperty("Remote Source")
// If you configured Maximum Cursor Size at the buscomp,
// get max-rows property
var maxRows = Inputs.GetProperty("max-rows");
// get search-string
var searchString = Inputs.GetProperty("search-string");
if (searchString != "" )
{
// convert the search-string into a where clause
searchString = stringReplace(searchString, '*', '%');
searchString = stringReplace(searchString, '[', ' ');
searchString = stringReplace(searchString, ']', ' ');
searchString = stringReplace(searchString, '~', ' ');
searchString = stringReplace(searchString, '"', "'");
whereClause = " where ";
whereClause = whereClause + searchString;
}
// match, search-spec, sort-spec
var childCount = Inputs.GetChildCount();
var child, sortProp;
for (var i = 0; i < childCount; i++)
{
child = Inputs.GetChild(i);
if (child.GetType() == "")
{
// Use this child property set if you want to use the old match field list.
// We are not using this in this example. We'll use search-string instead.
}
else if (child.GetType() == "search-spec")
{
// Use this child property set if you want to use the hierarchical
// representation of the search-string.
// We are not using this in this example. We'll use search-string instead.
}
else if (child.GetType() == "sort-spec")
{
// This child property set has the sort spec. We'll use this in this example
orderbyClause = " order by ";
var sortFieldCount = child.GetChildCount();
for (var j = 0; j < sortFieldCount; j++)
{
// Compose the order by clause.
sortProp = child.GetChild(j);
orderbyClause += sortProp.GetProperty("field");
var sortOrder = sortProp.GetValue();
if (sortOrder == "DESCENDING")
orderbyClause += " desc";
if (j < sortFieldCount-1)
orderbyClause += ", ";
}
}
}
// Now, our complete select statement is...
selectStmt += whereClause + orderbyClause;
// Now, query the data source.
var conn = getConnection();
var rs = getRecordset();
rs.Open(selectStmt, conn);
// We will return no more than maxRows of records.
var count = rs.RecordCount;
if (maxRows != "")
```

```
if (count > maxRows)
count = maxRows
// Iterate through the record set and add them to the Outputs PropertySet.
var fcount, fields, row;
for (i = 0; i < count; i++)
{
row = TheApplication().NewPropertySet();
fields = rs.Fields();
fcount = fields.Count;
for (j = 0; j < fcount; j++)
{
var fieldValue = fields.Item(j).Value;
if (fieldValue == null)
row.SetProperty(fields.Item(j).Name, "");
else
row.SetProperty(fields.Item(j).Name, fieldValue);
}
Outputs.AddChild(row);
rs.MoveNext();
}
// For debugging purposes...
logPropSet(Outputs, "QueryOutputs.xml" );
// clean up
child = null;
sortProp = null;
row = null;
rs.Close();
rs = null;
conn.Close();
conn = null;
return (CancelOperation);
}
function PreInsert (Inputs, Outputs) {
// For debugging purposes...
logPropSet(Inputs, "PreInsertInputs.xml");
var defaults = TheApplication().NewPropertySet();
defaults.SetProperty("Location", "KO");
Outputs.AddChild(defaults);
// For debugging purposes...
logPropSet(Outputs, "PreInsertOutputs.xml");
// Cleanup
defaults = null;
return (CancelOperation);
}
function Insert (Inputs, Outputs) {
// For debugging purposes...
logPropSet(Inputs, "InsertInputs.xml");
var fieldList = "";
var valueList = "";
// Inputs must have only 1 child property set.
var child = Inputs.GetChild(0);
var fieldName = child.GetFirstProperty();
var fieldValue;
while (fieldName != "")
{
fieldValue = child.GetProperty(fieldName);
if (fieldValue != "")
{
if (fieldList != "")
{
fieldList += ", ";
valueList += ", ";
}
fieldList += fieldName;
valueList += "'" + fieldValue + "'";
}
```

```
fieldName = child.GetNextProperty();
}
// The insert statement is...
var insertStmt = "insert into Contacts (" + fieldList + ") values (" + valueList + ")";
// Now, inserting into the data source...
var conn = getConnection();
conn.Execute (insertStmt);
// In this example, we must query back the record just inserted to get
// the value of its primary key. We made this primary key part of the buscomp
// to make update and delete easy. The primary key is "AccessId".
var selectStmt = "select * from Contacts where ";
var whereClause = "";
child = Inputs.GetChild(0)
fieldName = child.GetFirstProperty();
while (fieldName != "")
{
fieldValue = child.GetProperty(fieldName);
if (fieldName != "AccessId")
{
if (whereClause != "")
whereClause += " and ";
if (fieldValue == "")
whereClause += fieldName + " is null";
else
whereClause += fieldName + "='" + fieldValue + "'";
}
fieldName = child.GetNextProperty();
}
// The select statement is...
selectStmt += whereClause;
// Now, let's select the new record back
var rs = getRecordset();
rs.Open(selectStmt, conn);
// We're expecting only one row back in this example.
var fcount, fields, row, fieldValue;
row = TheApplication().NewPropertySet();
fields = rs.Fields();
fcount = fields.Count();
for (var j = 0; j < fcount; j++)
{
fieldValue = fields.Item(j).Value();
if (fieldValue == null)
row.SetProperty(fields.Item(j).Name(), "");
else
row.SetProperty(fields.Item(j).Name(), fieldValue);
}
Outputs.AddChild(row);
// For debugging purpose...
logPropSet(Outputs, "InsertOutputs.xml");
// Cleanup
child = null;
row = null;
rs.Close();
rs = null;
conn.Close();
conn = null;
return (CancelOperation);
}
function Update (Inputs, Outputs) {
// For debugging purposes...
logPropSet(Inputs, "UpdateInputs.xml");
var child;
var childCount = Inputs.GetChildCount();
var fieldName, fieldValue;
var updateStmt = "update Contacts set ";
var setClause = "";
```

```
var whereClause;
// Go through each child in Inputs and construct the
// necessary sql statements for update and query
for (var i = 0; i < childCount; i++)
{
child = Inputs.GetChild(i);
fieldName = child.GetProperty("Field Name");
fieldValue = child.GetProperty("Field Value");
// We only have to update changed fields.
if (child.GetProperty("Changed") == "true")
{
if (setClause != "")
setClause += ", ";
if (fieldValue == "")
setClause += fieldName + "=null";
else
setClause += fieldName + "='" + fieldValue + "'";
}
if (fieldName == "AccessId")
whereClause = " where AccessId = " + fieldValue;
}
// The update statement is...
updateStmt += setClause + whereClause;
// Now, updating the data source...
var conn = getConnection();
conn.Execute (updateStmt);
// How to construct the Outputs PropertySet can vary, but in this example
// We'll query back the updated record from the data source.
var selectStmt = "select * from Contacts" + whereClause;
// Now, let's select the updated record back
var rs = getRecordset();
rs.Open(selectStmt, conn);
// We expect only one row back in this example.
// In this example, we're returning all the fields and not just
// the updated fields. You can only return those updated
// fields with the new value in the Outputs property set.
var fcount, fields, row, fieldValue;
row = TheApplication().NewPropertySet();
fields = rs.Fields();
fcount = fields.Count();
for (var j = 0; j < fcount; j++)
{
fieldValue = fields.Item(j).Value();
if (fieldValue == null)
row.SetProperty(fields.Item(j).Name(), "");
else
row.SetProperty(fields.Item(j).Name(), fieldValue);
}
Outputs.AddChild(row);
// For debugging purposes...
logPropSet(Outputs, "UpdateOutputs.xml");
// Cleanup
child = null;
row = null;
rs.Close();
rs = null;
conn.Close();
conn = null;
return (CancelOperation);
}
function Delete (Inputs, Outputs) {
// For debugging purposes...
logPropSet(Inputs, "DeleteInputs.xml");
// Inputs must have only 1 child property set.
var child = Inputs.GetChild(0);
// In this example, we're only using the AccessId
```

```
    // (it is the primary key in the Contacts db)
    // for the delete statement for simplicity.
    var deleteStmt = "delete from Contacts where AccessId = " +
   child.GetProperty("AccessId");
    // Now, delete the record from the data source.
    var conn = getConnection();
    conn.Execute(deleteStmt);
    // For debugging purposes...
    logPropSet(Outputs, "DeleteOutputs.xml");
    // Returning empty Outputs property set.
    // clean up
    conn.Close();
    conn = null;
    return (CancelOperation);
    }
```

The following functions are helper functions:

```
    function getConnection () {
    // VBCContact is the ODBC data source name
    var connectionString = "DSN=VBCContact";
    var uid = "";
    var passwd = "";
    var conn = COMCreateObject("ADODB.Connection");
    conn.Mode = 3;
    conn.CursorLocation = 3;
    conn.Open(connectionString , uid, passwd);
    return conn;
    }
    function getRecordset() {
    var rs = COMCreateObject("ADODB.Recordset");
    return rs;
    }
    function logPropSet(inputPS, fileName) {
    // Use EAI XML Write to File business service to write
    // inputPS property set to fileName file in c:\temp directory.
    var fileSvc = TheApplication().GetService("EAI XML Write to File");
    var outPS = TheApplication().NewPropertySet();
    var fileLoc = "c:\\temp\\" + fileName;
    var tmpProp = inputPS.Copy();
    tmpProp.SetProperty("FileName", fileLoc);
    fileSvc.InvokeMethod("WritePropSet", tmpProp, outPS);
    // clean up
    outPS = null;
    fileSvc = null;
    tmpProp = null;
    }
    function stringReplace (string, from, to) {
    // Replaces from with to in string
    var stringLength = string.length;
    var fromLength = from.length;
    if ((stringLength == 0) || (fromLength == 0))
    return string;
    var fromIndex = string.indexOf(from);
    if (fromIndex < 0)
    return string;
    var newString = string.substring(0, fromIndex) + to;
    if ((fromIndex + fromLength) < stringLength)
    newString += stringReplace(string.substring(fromIndex+fromLength, stringLength),
   from, to);
    return newString;
    }
```

Siebel VB Business Service Example for a VBC

The following is an example of Siebel VB implementation of a business service for a VBC. The fields configured for this simple VBC are AccountId, Name, Phone, and Location.

CAUTION: Do not use Siebel CRM system fields, such as Id, as output properties. Undesired application behavior might result.

```
(declarations)
Option Explicit
Declare Function stringReplace(mystr As String, fromchar As String, tochar As
String) As String
Declare Function getData(execSQL As String, Results As PropertySet) As Integer
Function getData(execSQL As String, Results As PropertySet) As Integer
Dim sSrv As String, sDbn As String
Dim sUsr As String, sPsw As String
Dim oCon As Object, oRec As Object
Dim Row As PropertySet
Dim FileName, TextToSave
' *** SQL Server connectivity parameters
sSrv = "v817.siebel.com" '*** Oracle tns
sUsr = "system" '*** SQL Server: a user's login Id
sPsw = "manager" '*** SQL Server: a user's password
' *** Create SQL Server ADODB connection dynamically
Set oCon = CreateObject("ADODB.Connection")
oCon.Open "Provider=MSDAORA;" & _
"Data Source=" & sSrv & ";" & _
"User ID=" & sUsr & ";" & "Password=" & sPsw & ";"
' *** Perform SQL query
Set oRec = oCon.Execute(execSQL)
' *** Process SQL query result and save into file
While Not oRec.Eof
 Set Row=TheApplication.NewPropertySet()
 Row.SetProperty "AccountId", oRec.Fields.Item("AccountId").Value
 Row.SetProperty "Name", oRec.Fields.Item("Name").Value
 Row.SetProperty "Location", oRec.Fields.Item("Location").Value
 Row.SetProperty "Phone", oRec.Fields.Item("Phone").Value
 Results.AddChild Row
 Set Row = Nothing
 oRec.MoveNext
Wend
' *** Object cleanup
Set oRec = Nothing
Set oCon = Nothing
getData = 0
End Function
Sub Init(Inputs As PropertySet, Outputs As PropertySet)
Outputs.SetProperty "AccountId", ""
Outputs.SetProperty "Name", ""
Outputs.SetProperty "Phone", ""
Outputs.SetProperty "Location", ""
End Sub
Sub Query(Inputs As PropertySet, Outputs As PropertySet)
Dim sselectStmt As String
Dim swhereClause As String
Dim sorderbyClause As String
Dim ssearchstring As String
Dim child As PropertySet
Dim sortProp As PropertySet
Dim childCount As Integer
```

```
Dim i As Integer
Dim ret As Integer
Dim FileName, TextToSave
sselectStmt = "select * from siebel.Contact2 "
swhereClause = "where "
sorderbyClause= "order by "
ssearchstring = Inputs.GetProperty("search-string")
If Len(ssearchstring) > 0 Then
 ssearchstring = stringReplace(ssearchString, "*", "%")
 ssearchstring = stringReplace(ssearchString, "[", " ")
 ssearchstring = stringReplace(ssearchString, "]", " ")
 ssearchstring = stringReplace(ssearchString, "~", " ")
 ssearchstring = stringReplace(ssearchString, chr$(34), "'")
 sselectStmt = sselectStmt & swhereClause & ssearchstring
End If
' Write select statement to this file
FileName = "C:\Test.txt"
TextToSave = "select is " & sselectStmt
Open FileName For Append As #1
Print #1, TextToSave
Close #1
ret = getData(sselectStmt, Outputs)
End Sub
Function stringReplace(mystr As String, fromchar As String, tochar As String) As
String
'Replace all occurrences of fromchar in mystr with tochar
Dim i As Long
If Len(mystr) = 0 Or Len(fromchar) = 0 Then
stringReplace = mystr
Else
i = InStr(1, mystr, fromchar)
Do While i > 0
mystr = Left(mystr, i - 1) & tochar & Mid(mystr, i + Len(fromchar))
i = i + Len(fromchar)
i = InStr(i, mystr, fromchar)
Loop
stringReplace = mystr
End If
End Function
Function Service_PreInvokeMethod (MethodName As String, Inputs As PropertySet,
Outputs As PropertySet) As Integer
Service_PreInvokeMethod = ContinueOperation
If MethodName = "Init" Then
Service_PreInvokeMethod = CancelOperation
Init Inputs, Outputs
Exit Function
End If
If MethodName = "Query" Then
Service_PreInvokeMethod = CancelOperation
Query Inputs, Outputs
Exit Function
End If
End Function
```

9 Siebel EAI and File Attachments

Siebel EAI and File Attachments

Siebel EAI supports file attachments for exchanging business documents such as sales literature, activity attachments, and product defect attachments with another Siebel instance or an external system such as Oracle Applications. It includes the following topics:

- *About File Attachments*
- *Exchanging Attachments with External Applications*
- *Using MIME Messages to Exchange Attachments*
- *About the EAI MIME Hierarchy Converter*
- *About the EAI MIME Doc Converter*
- *Using Inline XML to Exchange Attachments*

About File Attachments

For example, if you are exchanging service requests with another application or partner, then you can include attachments such as screen captures, email, log files, and contract agreements that are associated with the service request in the information being exchanged. Siebel EAI support for file attachments allows comprehensive integration.

To use file attachments you first must create integration objects. For information, see *Integration Objects*, and *Creating and Maintaining Integration Objects*.

Siebel EAI offers the choice of integrating file attachments using MIME (the industry standard for exchanging multipart messages), or including the attachment within the body of the XML document, referred to as an inline XML attachment. Consider using inline XML attachments when integrating two instances of Siebel CRM using file attachments.

Exchanging Attachments with External Applications

Siebel EAI supports bidirectional attachment exchange with external applications using the following two message types:

- **MIME (Multipurpose Internet Mail Extensions).** MIME is the industry standard for exchanging multipart messages. The first part of the MIME message is an XML document representing the business object being exchanged and attachments to the object are included as separate parts of the multipart message. MIME is the recommended choice for integrating Siebel CRM with other applications. For more information, see *Using MIME Messages to Exchange Attachments*.
- **Inline XML attachments (Inline Extensible Markup Language).** With inline XML attachments, the entire business object you are exchanging, including any attachments, is sent as a single XML file. Consider using inline XML attachments when integrating two instances of Siebel CRM using file attachments. For more information, see *Using Inline XML to Exchange Attachments*.

Using MIME Messages to Exchange Attachments

To send or receive file attachments using MIME messages, Siebel EAI uses the MIME Hierarchy Converter and MIME Doc Converter.

You must perform the following procedures to use MIME to exchange attachments between Siebel CRM and another external system:

- Create an attachment integration object using the EAI Siebel Wizard business service.

  For information, see *Creating an Attachment Integration Object*.
- Create an inbound or outbound workflow.

  For information, see *Creating Workflow Examples*.
- Test your workflow using the Workflow Simulator.

  For information, see *Business Processes and Rules: Siebel Enterprise Application Integration* .

Creating an Attachment Integration Object

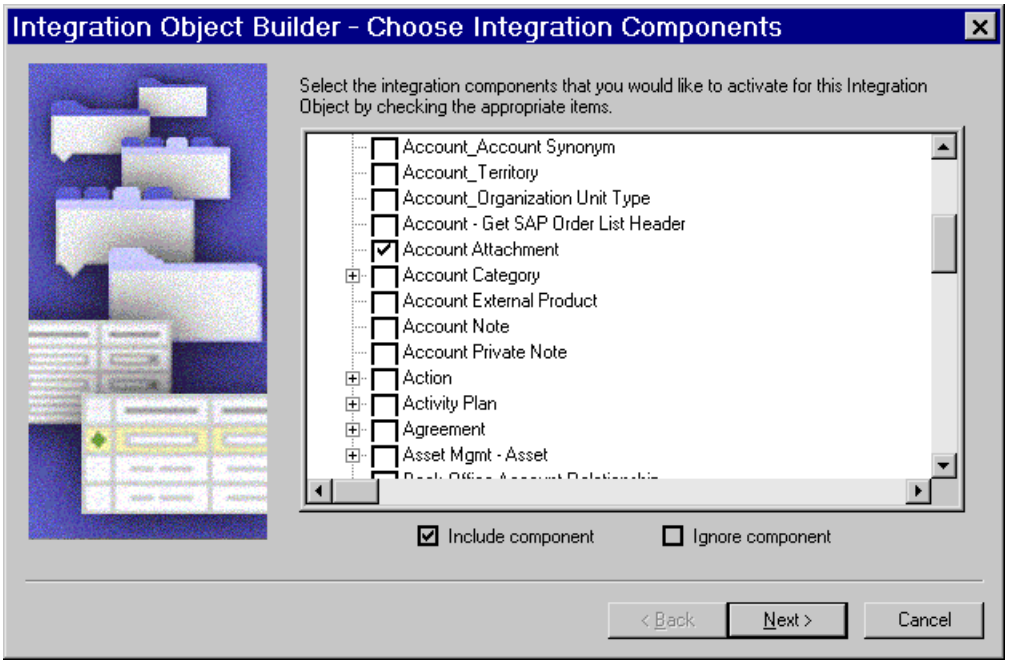
The following procedure guides you through the steps of creating an attachment integration object.

To create a new attachment integration object

1. In Siebel Tools or Web Tools, create a new workspace.
2. From the File menu, choose New Object to display the New Object Wizards dialog box.
3. Select the EAI tab, and then double-click Integration Object.

   The Integration Object Builder wizard appears.
4. Follow the procedure in *Creating Integration Objects Using the EAI Siebel Wizard Business Service* to create the new integration object, for example *SourceObject* Attachment.

   > **Note:** When creating your integration object you must select the Attachment integration component. The following figure illustrates this when the source object is Account.



5. In the Object Explorer, select Integration Object, and then select your new integration object in the Object List Editor.
6. In the Object Explorer, expand the Integration Object tree to show the Integration Component object.
7. Select the *SourceObject* Attachment integration component, and set its External Sequence and XML Sequence properties so that they are greater than those of the other integration components (that is, last in sequence), if not already set.

   If they are not last, the situation can arise where the attachment is processed successfully (and the file system is physically updated). Then a subsequent integration component causes a failure (for example, an attempt to insert to the database causes a duplicate error). In this case, the database transaction is rolled back, but the file system is not restored.
8. With the *SourceObject* Attachment integration component selected, expand the Integration Component object, and then select the Integration Component Field object.

   The Integration Components and Integration Component Fields lists appear.
9. Inactivate all integration component fields except the following:
   - *SourceObject* Attachment Id, for example, Accnt Attachment Id
   - *SourceObject*FileExt, for example, AccntFileExt
   - *SourceObject*FileName, for example, AccntFileName
   - Comment
10. Select the *SourceObject* Attachment Id component field, and then verify that its Data Type property is set to `DTYPE_ATTACHMENT`.
11. Deliver the workspace.

Creating Workflow Examples

Depending on whether you are preparing for an outbound or an inbound attachment exchange, design different workflows as described in the following two procedures.

For more information on creating workflows, see *Siebel Business Process Framework: Workflow Guide* .

Outbound Workflow

To process the attachment for an outbound request you must create a workflow to query the database, convert the integration object and its attachments into a MIME hierarchy, and then create a MIME document to send to the File Transport business service.

To create an outbound workflow

1. In Siebel Tools, select the Workflow Process object in the Object Explorer.
2. Right-click, then choose New Record.
3. Give the new workflow a name and associate it with a locked project.
4. Right-click, and then choose Edit Workflow Process.

   The Workflow Process Designer appears.
5. Create a workflow consisting of Start, End, and four Business Services. Set up each Business Service according to the task it must accomplish.
6. Define your process properties.

   Set process properties when you need global properties for the entire workflow.

Name	Data Type	Default String
SiebelMessage	Hierarchy	Leave blank.
Error Message	String	Leave blank.
Error Code	String	Leave blank.
Object Id	String	Leave blank.
Process Instance Id	String	Leave blank.
Siebel Operation Object Id	String	Leave blank.
MIMEHierarchy	Hierarchy	Leave blank.
SearchSpec	String	[Account.Name] = 'Sample Account'

Name	Data Type	Default String
<Value>	String	Default output is binary.

7. The first business service queries the Account information from the database using the EAI Siebel Adapter business service with the Query method. This step requires the following input and output arguments.

Input Argument	Type	Value	Property Name	Property Data Type
Output Integration Object Name	Literal	Sample Account	not applicable	not applicable
SearchSpec	Process Property	not applicable	SearchSpec	String

Property Name	Type	Output Argument
SiebelMessage	Output Argument	Siebel Message

Note: For more information on using the EAI Siebel Adapter, see *EAI Siebel Adapter Business Service*.

8. The second business service in the workflow converts the Account integration object and its attachments to a MIME hierarchy using the EAI MIME Hierarchy Converter business service with the SiebelMessage to MIME Hierarchy method. This step requires the following input and output arguments.

Input Argument	Type	Property Name	Property Data Type
Siebel Message	Process Property	SiebelMessage	Hierarchy

Note: For more information on the EAI MIME Hierarchy Converter, see *About the EAI MIME Hierarchy Converter*.

9. The third business service of the workflow converts the MIME hierarchy to a document to be sent to File Transport business service. This step uses the EAI MIME Doc Converter business service with the MIME Hierarchy To MIME Doc method. This step requires the following input and output arguments.

Property Name	Type	Output Argument
MIMEHierarchy	Output Argument	MIME Hierarchy

Input Argument	Type	Property Name	Property Data Type
MIME Hierarchy	Process Property	MIMEHierarchy	Hierarchy

Property Name	Type	Output Argument
<Value>	Output Argument	MIME Message

Note: For more information on the EAI MIME Doc Converter, see *About the EAI MIME Doc Converter*.

10. For the final step, set up the last business service of the workflow to write the information into a file using the EAI File Transport business service with the Send method. This step requires the following input arguments.

Input Argument	Type	Value	Property Name	Property Data Type
Message Text	Process Property	not applicable	<Value>	String
File Name	Literal	c:\temp\account.txt	not applicable	not applicable

Note: For information on File Transport, see *Transports and Interfaces: Siebel Enterprise Application Integration* .

Inbound Workflow Example

To process the attachment for an inbound request, you must create a workflow to read the content from a file, convert the information into a Siebel Message, and send to the EAI Siebel Adapter to update the database accordingly.

Note: When passing the process property value for a workflow from an external application (or another business service) as the input property set, the corresponding property name in the input property set must be same name as the process property and is case sensitive.

To create an inbound workflow

1. In Siebel Tools, select the Workflow Process object in the Object Explorer.
2. Right-click, and then choose New Record.
3. Give the new workflow a name and associate it with a locked project.
4. Right-click, then choose Edit Workflow Process.
   The Workflow Process Designer appears.
5. Create a workflow consisting of Start, End and four Business Services. Set up each Business Service according to the task it must accomplish.
6. Define your process properties.

Set process properties when you need global properties for the entire workflow:

Name	Data Type
SiebelMessage	Hierarchy
Error Message	String
Error Code	String
Object Id	String
Siebel Operation Object Id	String
MIMEHierarchy	Hierarchy
MIMEMsg	Binary

7. The first business service in the workflow reads the Account information from a file using the EAI File Transport business service with Receive method. This step requires the following input and output arguments.

Input Argument	Type	Value
File Name	Literal	c:\temp\account.txt

Property Name	Type	Output Argument
<Value>	Output Argument	Message Text

> **Note:** For information on File Transport, see *Transports and Interfaces: Siebel Enterprise Application Integration* .

8. The second business service of the workflow converts the Account information to a MIME hierarchy using the EAI MIME Doc Converter business service with the MIME Doc to MIME Hierarchy method. This step requires the following input and output arguments.

Input Argument	Type	Property Name	Property Data Type
MIME Message	Process Property	<Value>	Binary

Input Argument	Type	Property Name	Property Data Type

Property Name	Type	Output Argument
MIMEHierarchy	Output Argument	MIME Hierarchy

9. The third business service of the workflow converts the MIME hierarchy to a document, and sends it to the EAI Siebel Adapter business service. This step uses the EAI MIME Hierarchy Converter business service with the MIME Hierarchy to Siebel Message method. This step requires the following input and output arguments.

Input Argument	Type	Property Name	Property Data Type
MIME Hierarchy	Process Property	MIMEHierarchy	Hierarchy

Property Name	Type	Output Argument
SiebelMessage	Output Argument	Siebel Message

10. The last step of the workflow writes the information into the database using the EAI Siebel Adapter business service with the Insert or Update method. This step requires the following input argument.

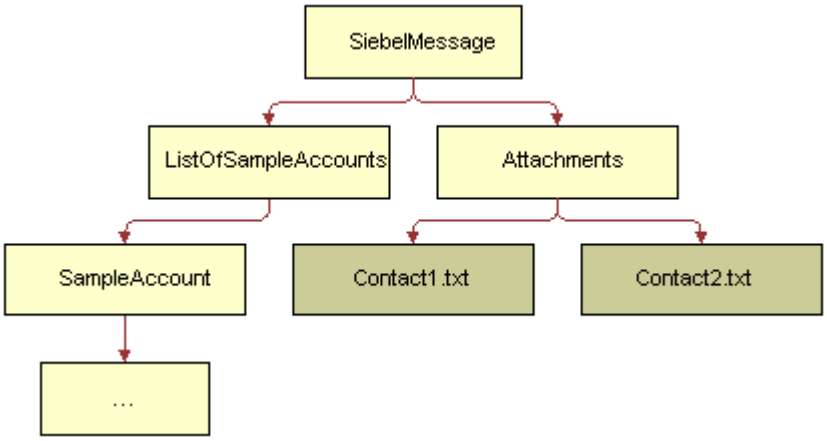
Input Argument	Type	Property Name	Property Data Type
Siebel Message	Process Property	SiebelMessage	Hierarchy

About the EAI MIME Hierarchy Converter

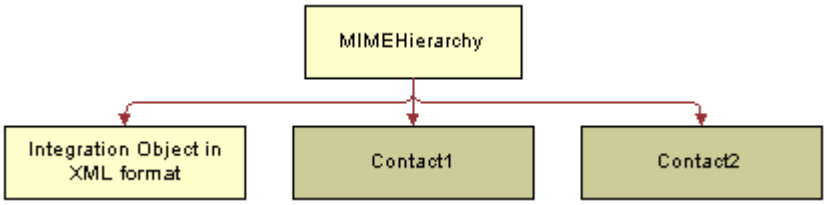
The EAI MIME Hierarchy Converter transforms the Siebel Message into a MIME (Multipurpose Internet Mail Extensions) hierarchy for outbound integration. For inbound integration, it transforms the MIME Hierarchy into a Siebel Message.

Outbound Integration

The EAI MIME Hierarchy Converter transforms the input Siebel Message into a MIME Hierarchy. The following figure illustrates the Siebel Message of a sample Account with attachments. This figure represents both input and output to the MIME Hierarchy Converter.



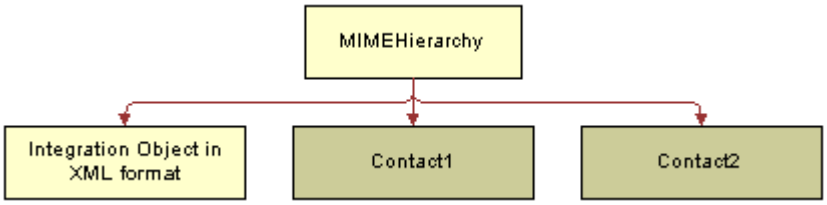
The output of this process is illustrated in the following figure.



The first child of a MIME Hierarchy is the XML format of the Sample Account Integration Object instance found in the Siebel Message. The remaining two children are the corresponding children found after Attachments. If there is no child of type Attachments in the Siebel Message, then the output is just a MIME Hierarchy with a child of type Document. This document will contain the XML format of the Sample Account integration object instance.

Inbound Integration

The MIME Hierarchy Converter transforms a MIME Hierarchy input into a Siebel Message. For the inbound process, the first child of the MIME Hierarchy has to be the XML format of the Integration Object instance; otherwise, an error is generated. The following figure illustrates the incoming hierarchy.



The output of this process is illustrated in the figure in the previous topic. The output for this process is the same as the input.

About the EAI MIME Doc Converter

The MIME Doc Converter converts a MIME Hierarchy into a MIME Message and a MIME Message into a MIME Hierarchy. A MIME Hierarchy consists of two different types of property sets, as shown in the following table.

Property	Description
MIME Hierarchy	Mapping to a MIME multi-part
Document	Mapping to MIME basic-part

EAI MIME Doc Converter Properties

The following table illustrates some examples of how a MIME Message maps to a MIME Hierarchy.

MIME Message	MIME Hierarchy
MIME-Version: 1.0 Content-Type: application/xml Content-Transfer-Encoding: 7bit This is a test.	Type: Document Value: This is a test
MIME-Version: 1.0 Content-Type: multipart/related; type="application/xml"; boundary=--abc ----abc Content-Type: application/xml Content-Transfer-Encoding: 7bit This is test2. ----abc--	Type: MIMEHierarchy Type: Document Value: This is a test2

The business service needs the following properties on the child property set as shown in the following table. These properties reflect the most accurate information on the data contained in the child property set.

Property	Possible Values	Type	Description
ContentId	Any value	Document	No Default. The ContentId is the value used to identify the file attachment when the receiver parses the MIME

Property	Possible Values	Type	Description
			message. When importing attachments, use a unique value for this property and not repeat it for the rest of the file attachments. This is required in the actual document. This property is automatically populated when you are exporting an attachment from a Siebel application.
Extension	txt, java, c, C, cc, CC, h, hxx, bat, rc, ini, cmd, awk, html, sh, ksh, pl, DIC, EXC, LOG, SCP, WT, mk, htm, xml, pdf, AIF, AIFC, AIFF, AU, SND, WAV. gif, jpg, jpeg, tif, XBM, avi, mpeg, ps, EPS, tar, zip, js, doc, nsc, ARC, ARJ, B64, BHX, GZ, HQX	Document	No Default. If ContentType and ContentSubType are not defined, then Extension is used to retrieve the appropriate values from this property. If all three values are specified, then ContentType and ContentSubType values override the values retrieved from the Extension. If either the Extension or both ContentType and ContentSubType are not specified, then ContentType will be set to application and ContentSubType will have the value of octet-stream.
ContentType	application, audio, image, text, video	Document	Default is application. The ContentType value has to be specified if you want to set the content type of the document instead of using the extension to get a value from the MIME utility function. If the value is not provided, then the default value is used. The ContentType of multipart is used to represent file attachments in a MIME message. Other values to describe a multipart message are not supported.
ContentSubType	plain, richtext, html, xml (used with ContentType of Text) octet-stream, pdf, postscript, x-tar, zip, x-javascript, msword, x-conference, x-gzip (used with ContentType of application) aiff, basic, wav (used with ContentType of audio) gif, jpeg, tiff, x-xbitmap (used with ContentType of image) avi, mpeg (used with ContentType of video)	Document	Default is octet-stream. The ContentSubType value has to be specified if you want to set the content subtype of the document instead of using the extension to get a value from the MIME utility function. If the value is not provided, then the default value is used. **Note:** Octet-stream is transparent and uses nonencoded 8-bit bytes. The MIME message will contain the binary file data as is, which might cause issues in data transmission over networks that remove bit number 8 (the hi-bit) for special needs.

Note: On the inbound direction, the business service is independent of the transport. It assumes that the input property set contains the MIME message, and writes a property set representation of the MIME message. A property set is used to represent each part of the MIME message. When decoding the MIME message, the business service automatically sets the properties based on the values in the MIME message.

Using Inline XML to Exchange Attachments

To exchange attachments between applications, you use the EAI Siebel Adapter business service:

- To send a message to an external application, call the EAI Siebel Adapter with an integration object that has an integration component from an attachment business component. The EAI Siebel Adapter generates the integration object hierarchy and then converts it to an XML document. The attachment is included in the XML in the *SourceObject*FileBuffer element.
- To insert an attachment into a Siebel CRM application, the external application uses the same integration object hierarchy, making sure the required fields are present, and puts the base64 string corresponding to the attachment into this message. The XML converter converts the message into an integration object hierarchy, and the EAI Siebel Adapter inserts the attachment.

Note: Attachments must be in base64 format.

Perform the following tasks to create and test inline XML attachments using an integration object and a workflow:

- Creating an attachment integration object using the EAI Siebel Wizard business service

  For information, see *Creating an Attachment Integration Object*.

  CAUTION: To avoid SQL errors, you must inactivate all integration component fields in the integration object except those in Step 9.

- *Creating an Attachment*
- *Creating a Test Workflow*
- Testing your workflow using the Workflow Simulator

  For information, see *Business Processes and Rules: Siebel Enterprise Application Integration* .

Creating an Attachment

You create an attachment to a record in the Siebel client whose row ID you know.

To create the attachment

1. In the Siebel client, navigate to a record that can take an attachment, for example, an account.
2. Choose Help, then About Record from the application-level menu to obtain the row ID of the record.
3. Drill down on the record, then select the Attachments tab.
4. Add an attachment to the record if none exists.

Creating a Test Workflow

You create a workflow in Siebel Tools to do the following:

- Query the Siebel database for the record with the attachment.

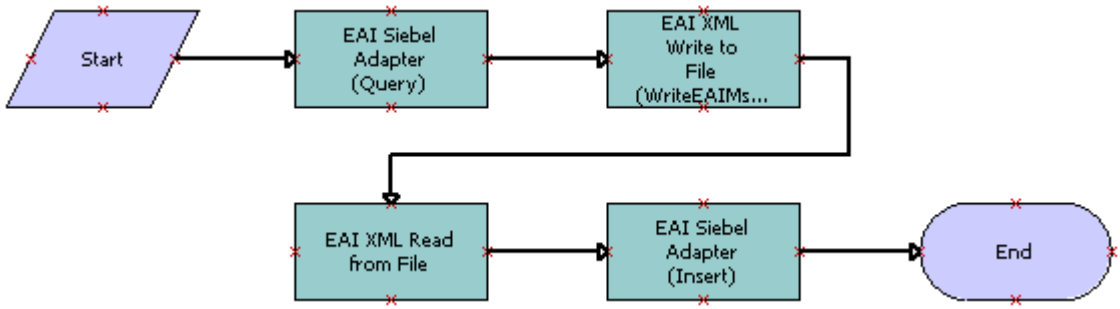
- Convert the integration object and its attachment into a Siebel Message.
- Read an external XML file (containing an attachment) and convert it into a Siebel Message.

  Note: The XML file must use the exact integration object hierarchy as the attachment integration object you created.

- Insert the record into the Siebel database.

To create a test workflow to exchange attachments

1. Create the following workflow:



2. Define the process properties as shown in the following table.

Name	Data Type
Error Code	String
Error Message	String
Object Id	String
Process Instance Id	String
Siebel Operation Object Id	String
SiebelMessage	Hierarchy

3. The first business service step queries the database using the EAI Siebel Adapter business service with the Query method. This step requires the following input and output arguments:

Input Argument	Type	Value
OutputIntObjectName	Literal	Attachment integration object you created, for example, Account Attachment

Input Argument	Type	Value
PrimaryRowId	Literal	Row ID of the record to which you added an attachment

Property Name	Type	Output Argument
SiebelMessage	Output Argument	SiebelMessage

4. The second business service step writes the integration object hierarchy to an XML file using the EAI XML Write to File business service with the WriteEAIMsg method. This step requires the following input arguments:

Input Argument	Type	Value
FileName	Literal	File to write, for example, d:\temp\AttachmentTest_write.xml
SiebelMessage	Process Property	SiebelMessage

5. The third business service step reads an XML hierarchy and converts it into a Siebel Message using the EAI XML Read From File business service with the ReadEAIMsg method. This step requires the following input and output arguments:

Input Argument	Type	Value
FileName	Literal	File to read, for example, d:\temp\AttachmentTest_read.xml **Note:** For testing purposes, you can use a modified form of the file written in the second business step, which will automatically have the correct hierarchy.

6. The fourth business service step reads the Siebel Message and inserts the record into the Siebel database using the EAI Siebel Adapter business service with the Insert method. This step requires the following input argument:

Property Name	Type	Output Argument
SiebelMessage	Output Argument	SiebelMessage

Input Argument	Type	Value
SiebelMessage	Process Property	SiebelMessage

10 External Business Components

External Business Components

The external business component feature provides a way to access data that resides in a non-Siebel table or view, using a Siebel business component.

This chapter consists of the following topics:

- *Process of Configuring External Business Components*
- *Using Specialized Business Component Methods for EBCs*
- *Usage and Restrictions for External Business Components*
- *About Using External Business Components with the Siebel Web Clients*
- *About Overriding Connection Pooling Parameters for the Data Source*
- *About Joins to Tables in External Data Sources*
- *About Distributed Joins*
- *Troubleshooting External Business Components*

Before continuing with configuring and implementing external business components (EBCs), review the following books on the *Siebel Bookshelf* :

- *Configuring Siebel Business Applications*
- *Siebel Developer's Reference*
- *Siebel Tools Online Help*
- *Using Siebel Tools*

Process of Configuring External Business Components

Before proceeding, review *Configuring the External Business Component*. To configure EBCs, you perform the following high-level tasks:

1. *Creating the External Table Definition*.
   Import the external table definition into Siebel Tools using the External Table Schema Import Wizard.
   This wizard creates a new Table object definition in the Siebel Repository, based upon the contents of a DDL (data definition language) file, or from an Open Database Connectivity (ODBC) data source.
2. *Mapping External Columns to Siebel CRM System Fields*.
   Map columns in the external table or view to Siebel CRM system fields.

   > **Note:** One column in the external table or view must be mapped to the Id system field by setting the System Field Mapping property for the column.

3. *Specifying the Data Source Object*.

Configure the table definition and specify the data source object.

The Data Source object is a child object of the Table Object in Siebel Tools and will have to be exposed in the Object Explorer if not already visible. For information on exposing objects in the Object Explorer of Siebel Tools, see *Using Siebel Tools* .

This object tells the object manager which data source to use to access the object.

4. *Specifying Any Optional Table Properties*.

   When the table is imported, specify additional table properties for the corresponding external table.
5. *Configuring the External Business Component*.

   Configure the EBC and specify the data source object. This data source name will be the same as that specified for the Table object.
6. *Specifying Run-Time Parameters*.

   After the data source definition is named in Siebel Tools, specify the run-time parameters by completing the following:
   - Configure the data source definition.
   - Update the server component definition.

Creating the External Table Definition

You use Siebel Tools and the External Table Schema Import Wizard to import your external table definition into the Siebel Repository.

You can create the table definition in one of two ways:

- *Creating the External Table Definition from a DDL File*
- *Creating the External Table Definition from an ODBC Data Source*

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

This task is a step in *Process of Configuring External Business Components*.

Creating the External Table Definition from a DDL File

You can use the External Table Schema Import Wizard to create the table definition from a data definition language (DDL) file.

It is possible to import an external view definition rather than a table definition. When a view rather than a table definition is imported, it is necessary to amend the Type property of the created Table definition to reflect *External View*.

> **Note:** You can import a database view definition as well as a table definition here. While no difference exists in the resulting Siebel Table object, if it references an external database view, only read access from the Siebel Application is supported.

To create the external table definition from a DDL file

1. In Siebel Tools, check out and lock the appropriate project.
2. From the File menu, choose New Object to display the New Object Wizards dialog box.
3. Click the General tab, then double-click External Table Schema Import.

4. In the External Table Schema Import Wizard, specify the following values, then click Next:
    a. Choose the project with which the new Table object definition will be associated.
    b. Specify DDL/Analytics as the schema source type.
    c. Choose the three-digit group code for table name generation. For example, if you choose AXA, then the format of the table names generated will be:

    ```
    EX_AXA_00000001
    ```
5. In the Import External Schema - DDL dialog box, specify the following, then click Next:
    a. Specify the database where the external table resides. The value specified must correspond to the database used by the Siebel schema, for example, Oracle Server Enterprise Edition.
    b. Provide the full path for the location of the SQL/DDL file that contains the external table definition.
6. Confirm the entries, then click Finish to import the DDL file.

    A Table object definition is added to the Siebel Repository, corresponding to the external table.
7. Repeat Step 2 through Step 6 for every external table definition you want to import.

Creating the External Table Definition from an ODBC Data Source

You can use the External Table Schema Import Wizard to create the table definition from an Open Database Connectivity (ODBC) data source.

To create the external table definition from an ODBC data source

1. In Siebel Tools, check out and lock the appropriate project.
2. From the File menu, choose New Object to display the New Object Wizards dialog box.
3. Click the General tab, then double-click External Table Schema Import.
4. In the External Table Schema Import Wizard, specify the following values, then click Next:
    a. Choose the project with which the new Table object definition will be associated.
    b. Specify ODBC as the schema source type.
    c. Choose the three-digit group code for table name generation. For example, if you choose AXA, then the format of the table names generated will be:

    ```
    EX_AXA_00000001
    ```
5. In the next dialog box, click Select Data Source.

    The Select Data Source dialog box appears.
6. Click the Machine Data Source tab, select the appropriate data source name, and then click OK.
7. In the Connect to *Database Type* dialog box, on the Login tab enter the database user ID and password, then click OK.
8. Enter the table owner for the data source, then click Next.
9. Select the tables to import, then click Next.
10. Confirm the entries, then click Finish.

    A Table object definition is added to the Siebel Repository for each external table selected.

About Data Type Mappings for Importing Table Definitions

When importing table definitions, certain data type mappings are supported for use with the Siebel application. The following table contains the data type mappings you can use when importing table definitions.

Supported Data Type	Siebel Data Type
Microsoft SQL Server Data Types	
int	Numeric with scale of 0
bigint	Numeric with scale of 0
smallint	Numeric with scale of 0
tinyint	Numeric with scale of 0
float	Numeric
real	Numeric
decimal	Numeric
money	Numeric
smallmoney	Numeric
bit	Numeric with a value of 0 or 1
char	Character
nchar	Character
varchar	Varchar
nvarchar	Varchar
text	Long
ntext	Long
datetime	Date Time
smalldatetime	Date Time
IBM DB2 UDB Data Types	
UINT	Numeric with scale of 0

Supported Data Type	Siebel Data Type
BIGUINT	Numeric with scale of 0
SMALLUINT	Numeric with scale of 0
FLOAT	Numeric
REAL	Numeric
DECIMAL	Numeric
NUMERIC	Numeric
CHAR	Character
VARGRAPHIC	Varchar
LONG VARGRAPHIC	Long
CLOB	CLOB
DATE	Datetime
TIME	Datetime
TIMESTAMP	Datetime
Oracle Database Data Types	
Number	Numeric
TIMESTAMP WITH TIME ZONE	Numeric
TIMESTAMP WITH LOCAL TIME ZONE	Numeric
Char	Character
Nchar	Character
varchar2	Varchar
nvarchar2	Varchar

Supported Data Type	Siebel Data Type
Long	Long
CLOB	CLOB
date	Datetime
Oracle Business Intelligence (BI) Server Data Types	
Integer	Numeric with scale of 0
Smallint	Numeric with scale of 0
Tinyint	Numeric with scale of 0
Float	Numeric
Double	Numeric
Bit	Character (1)
Boolean	Character (1)
Char	Character
Varchar	Varchar
Longvarchar	Long
Datetime	Datetime
Date	Datetime
Time	Datetime

The following table contains the data types that are not supported for importing table definitions.

Database	Unsupported Data Types
Microsoft SQL Server	timestamp
	varbinary

Database	Unsupported Data Types
	binary
	image
	cursor
	uniqueidentifier
IBM DB2 UDB	DBCLOB
	BLOB
Oracle Database	TIMESTAMP
	NCLOB
	BLOB
	BFILE
	ROWID
	UROWID
	RAW
	LONG RAW
	INTERVAL YEAR TO MONTH
	INTERVAL DAY TO SECOND
Oracle BI Server	Timestamp
	Varbinary
	Longvarbinary
	Binary
	Object

Database	Unsupported Data Types
	Unknown

About the New Imported Table Definition

After the table definition is imported using the External Table Schema Import Wizard, the external table and the external column names are generated.

The external table name is stored in the Table object's Alias property. This external table name consists of the following:

- An `EX` prefix (for external table)
- A three-digit batch code specified in the External Table Schema Import Wizard
- An automatically generated seven-digit number

An example of the Table name is `EX_ABC_0000001`.

The external column name is stored in the Column child object's Alias property. An `X` is added as the prefix and a four-digit number is added as the suffix for the external column name, for example, `X_ABC_0000001_0001`.

The Table object's Type property is set to External or External View (if a view was imported). This column denotes that the table resides outside of the Siebel database.

Mapping External Columns to Siebel CRM System Fields

This task is a step in *Process of Configuring External Business Components*.

When the EBC is defined, you must map the Siebel CRM system fields to the corresponding external table column. System field mapping is accomplished at the column level, rather than using business component user properties. Specify the System Field Mapping column attribute if you want to map a Siebel system field to a column.

> **Note:** At a minimum, the Id field must be mapped to a unique column defined in the external table and in the Table object definition, which is specified in the business component's Table property.

By default, the Siebel CRM system fields are not included in the generated SQL for external tables.

System Field Mapping is used to specify the mapping between table columns and Siebel CRM system fields. The following is a list of the system fields that can be mapped to external table columns:

- **Conflict Id.** (Optional).
- **Created.** (Optional) Datetime corresponding to when the record was created.
- **Created By.** (Optional) String containing the user name of the person and the system that created the records.
- **Extension Parent Id.** (Optional).
- **Mod Id.** (Optional).
- **Non-system.** (Optional).
- **Updated.** (Optional) Datetime corresponding to when the record was last updated.
- **Updated By.** (Optional) String containing the user name of the person and system that last updated the record.

- **Id.** Mandatory. The single column unique identifier of the record. A column in the external table must be mapped to the Id field.

Note: The System Field Mapping property must be used in conjunction with external tables only.

Specifying the Data Source Object

This task is a step in *Process of Configuring External Business Components*.

When the external table has been defined, specify the data source for the corresponding external table. The Data Source child object of the Table object specifies the data source for the corresponding external table:

- The Data Source child object corresponds to a data source defined in the application configuration file (.cfg) or in the Application - Server Configuration screen, Profile Configuration view.
- The Data Source child object instructs the Application Object Manager to use the data source for a specific table. If a Data Source child object is not specified, then the default data source for the application will be used.

Note: The Data Source child object is specified for external tables only.

For more information about the Data Source child object, see *Siebel Tools Online Help* .

Specifying Any Optional Table Properties

When the table is imported, you can specify additional table properties for the corresponding external table:

- **External API Write.** Allows you to perform reads directly from the database and have write operations processed by way of a script.

  A Boolean property is used to indicate whether or not inserts, updates, or deletes to external tables will be handled by an external API. If this property is set to TRUE, then add scripts to the BusComp_PreWriteRecord and BusComp_PreDeleteRecord events to publish the insert, update, or delete operation to an external API.
- **Key Generation Business Service.** Allows a business service to generate a primary key (Id field) for a business component. If this is not specified, then the Siebel application will generate a row_id value for the column that corresponds to the Id system field.
- **Key Generation Service Method.** Allows a business service method to be called when generating a primary key for a business component.

  This method returns a property with the name set to the external table's key column name, and the value set to the generated key:

  ```
  Outputs.SetProperty(<my_external_key_column_name>, <generated_key>);
  ```

For more information about these table properties, see *Siebel Tools Online Help* .

This task is a step in *Process of Configuring External Business Components*.

Configuring the External Business Component

When a Table object definition corresponding to the external table exists in the repository, you can configure a business component to use the new Table object definition.

In general, configuring an EBC is similar to configuring a standard business component with the following exceptions:

- **Data Source business component property.** Specify the Data Source business component property. Set the value for this property to the name of the corresponding Table Data Source.
- **Log Changes property.** Set the Log Changes property to False (unchecked). This will prevent Siebel Remote or Replication transactions from being created. (The default is true.)
- **Intersection table.** When configuring a many-to-many relationship, the intersection table resides in the same database instance as the child table.
- **CSSBusComp class.** It is recommended that all EBCs use the CSSBusComp class.

Note: Substituting a Siebel-provided table with an external table can result in significant downstream configuration work, and in some cases can restrict or prevent the use of standard functionality provided for Siebel CRM.

This task is a step in *Process of Configuring External Business Components*.

Specifying Run-Time Parameters

After the data source definition is named in Siebel Tools, you specify the run-time parameters by configuring the data source definition, and updating the server component definition.

If you are testing by using the Siebel Developer Web Client, then add a [DataSource] section to the client .cfg file.

This task is a step in *Process of Configuring External Business Components*.

Configuring the Data Source Definition

As part of specifying the run-time parameters, configure the data source definition.

To configure the data source definition

1. Navigate to the Administration - Server Configuration screen, Enterprises view, then Profile Configuration.
2. Copy an existing InfraDatasources named subsystem type.
3. Change the Profile and Alias properties to the Data Source name configured in Siebel Tools.
4. Update the profile parameters to correspond to the external RDBMS:
    - DSConnectString = *data source connect string*
        - For Microsoft SQL Server or IBM DB2, create an ODBC or equivalent connection and input the name of this in the parameter.
        - For Oracle Database, this value must specify the TNS name associated with the database and not an ODBC or other entry.
    - DSSQLStyle = *database SQL type*

See the following table for a listing of the supported SQL types.

- DSDLLName = *DLL or library name corresponding to the SQL type*

  See the following table for a listing of the supported connector Dynamic Link Library (DLL, Windows) or library (shared object, UNIX) names and SQL styles.
- DSTableOwner = *data source table owner*
- DSUsername = *default username used for connections* (Optional)
- DSPassword = *default password used for connections* (Optional)

> **Note:** The DSUsername and the DSPassword parameters are optional. However, to avoid receiving a login prompt when accessing the external data source, specify *DSUsername* and *DSPassword*. If specified, they will override the default username and password.

The DSUsername and the DSPassword parameters are activated only when using the Database Security Adapter. For more information, see *Configuring a User in LDAP or ADSI Security Adapter To Access EBCs*.

Configuring a User in LDAP or ADSI Security Adapter To Access EBCs

You do not typically use Lightweight Directory Access Protocol (LDAP) or Microsoft Active Directory Service Interfaces (ADSI) to retrieve the database username and password to be used for the EBC. Instead, you specify a common DSUsername and DSPassword as described in *Configuring the Data Source Definition*.

> **Note:** You do not need to configure LDAP or ADSI authentication unless you need different database users for the EBC (more than one for all users), for example, to have all call center users access the EBC data source with a specific database user, or to be able to set a specific database user for one or more Siebel login IDs.

When LDAP or ADSI authentication is used, the username and password values for the external data source are provided in the ADSI `SharedCredentialsDN` parameter and the `CredentialAttributeType` attribute.

> **Note:** The `CredentialAttributeType` attribute must be able to hold multiple values, for example, `url`. It cannot be single-value, such as `mail`.

For example, the name of your external data source is *MyExtDataSrc*, and your ADSI is configured with the following parameters:

```
SharedCredentialsDN= cn=sharedcred,ou=people,dc=siebel,dc=com
CredentialAttributeType = url
```

In your ADSI server modify the `url` attribute for the following entry:

```
cn=sharedcred,ou=people,dc=siebel,dc=com
```

Before modifying, one value must already exist (assuming *sadmin* and *db2* are the user name and password for the *ServerDataSrc* data source, which is the primary data source):

```
type=ServerDataSrc username=sadmin password=db2
```

Add additional values to the `url` attribute (assuming *mmay* and *mmay* are the user name and password for the *MyExtDataSrc* data source, which is the external data source):

```
type=MyExtDataSrc username=mmay password=mmay
```

After adding the new value for the external data source to the `url` attribute, you are able to access EBCs.

Configuring the Data Source Definition for the Siebel Developer Web Client

If testing using the Siebel Developer Web Client, then add a [DataSource] section to the client .cfg file for the data source definition named in Siebel Tools. In this example, WindyCity is the data source being added.

To configure the data source definition in the Siebel Developer Web Client

1. Add the data source definition named in Siebel Tools. In this example, the data source definition named is WindyCity:

```
[DataSources]
Local = Local
Sample = Sample
ServerDataSrc = Server
GatewayDataSrc = Gateway
WindyCity = WindyCity
```

2. In the data source section of the application's .cfg file, add the following parameters (for the supported SQL types and connector DLL names, see the following table):

```
[WindyCity]
Docked = TRUE
ConnectString = data source connect string
SqlStyle = database SQL type
TableOwner = data source table owner

 DLL =
 DLL Name corresponding to the SQL type
DSUsername = user id (Optional)
DSPassword = password (Optional)
```

Supported Connector Names and SQL Styles

When defining the DLL (Windows) or library (UNIX) and SQL files for importing the external schema, the external database being used might not be the same as the Siebel database. The following table contains the supported connector DLL and library names and their corresponding SQL styles. The extension for the DLL or library name is optional.

External Database	DLL Name (Windows)	Library Name (UNIX)	SQL Style
IBM DB2	sscddcli.dll	sscddcli.so	DB2
Microsoft SQL Server	sscdms80.dll	Not supported	MSSqlServer
Oracle Database	sscdo90.dll	sscdo90.so	OracleCBO

Updating the Server Component to Use the New Data Source

As part of specifying the run-time parameters, update the server component to use the new data source.

To update the server component to use the new data source

1. Navigate to the Administration - Server Configuration screen, Enterprises view, then Component Definitions.
2. In the Component Definitions list applet, select your Application Object Manager Component. For example, select the Call Center Object Manager (ENU).
3. Choose Start Reconfiguration from the Menu drop-down list on the Component Definitions list applet.

   The Definition State of the component will be set to Reconfiguring. Reselect your application component after selecting the Start Reconfiguring menu item.
4. In the Component Parameters list applet, query for OM - Named Data Source name, and update the Value by adding the alias name of the data source specified in *Specifying Run-Time Parameters*.

   The format of the OM - Named Data Source name parameter is a comma-delimited list of data source aliases. It is recommended that you do not modify the default values, and that you add their new data sources to the preexisting list.
5. After the parameter values are reconfigured, commit the new configuration by selecting Commit Reconfiguration from the Menu drop-down list on the Component Definitions list applet.

   The new parameter values are merged at the enterprise level.

   To cancel the reconfiguration before it has been committed, select Cancel Reconfiguration from the Menu drop-down list on the Component Definitions list applet.

Using Specialized Business Component Methods for EBCs

The following are the specialized business component methods that are supported for use with EBCs:

- IsNewRecordPending
- GetOldFieldValue
- SetRequeryOnWriteFlag (PreWriteRecord event)
- SetRequeryOnWriteFlag (WriteRecord event)

IsNewRecordPending Business Component Method

This method can be called by using a script in the PreWriteRecord event to determine if the current record is newly created. If the record is a new record, then this method returns the value TRUE.

An example script for the use of this method follows:

```
var isNewRecord = this.InvokeMethod("IsNewRecordPending");
```

GetOldFieldValue Business Component Method

This method can be called by using a script in the PreWriteRecord event to retrieve an old field value if needed. This method takes an input parameter, which must be a valid field name, and returns a string containing the old field value.

An example script for the use of this method follows:

```
var oldLoc = this.InvokeMethod("GetOldFieldValue", "Location");
```

SetRequeryOnWriteFlag (PreWriteRecord event) Business Component Method

In the PreWriteRecord event, this method can be used to put the business component into a mode where the current record refreshes from the data source after the write operation. EBCs typically use this method to refresh database sequencing column values on new record operations. This method takes an input parameter of TRUE or FALSE.

An example script for the use of this method follows:

```
var requery = this.InvokeMethod("SetRequeryOnWriteFlag", "TRUE");
```

SetRequeryOnWriteFlag (WriteRecord event) Business Component Method

In the WriteRecord event, this method informs the object manager that the write operation to the data source is processed by using a script rather than a database connector. At the end of the operation, the business component method, SetRequeryOnWriteFlag, can be called again with the FALSE parameter to reset the requery on write mode, if needed.

An example script for the use of this method follows:

```
var extWrite = this.InvokeMethod("SetRequeryOnWriteFlag", "TRUE");
// insert script here to commit the record via an mechanism channel
var resetWrite = this.InvokeMethod("SetRequeryOnWriteFlag", "FALSE");
```

Usage and Restrictions for External Business Components

The following usage guidelines and restrictions apply to EBCs:

- Creating and populating the external table is the responsibility of the customer. Consult your database administrator for the appropriate method to use.
- EBCs cannot be docked, so they do not apply to mobile users on the Siebel Mobile Web Client. Siebel Remote is not supported.
- EBCs support many-to-many relationships with the limitation that for such relationships the intersection table must be from the same data source as the child business component.
- EBCs cannot be loaded using the Enterprise Integration Manager.
- EBCs rely on the Business Object Layer of the Siebel Architecture. Therefore, EBCs are used only in Siebel Server components using this layer such as the Application Object Manager (for example, the Call Center

Object Manager), Workflow Process Manager, and so on. EBCs are not used on components not using this layer, such as Workflow Policies (the Workflow Monitor agent) and Assignment Manager.

- The Id field must be mapped to an underlying column in the external table to support insert, update, delete, and query operations.
- Using the Oracle Sequence Object to populate the Id system field is not supported. The value of the Id system field has to be known by the object manager at the record commit time, while the Oracle Sequence Object value is populated by the database server when the change is being processed inside the database.
- If the column that was mapped to the Id system field has *Primary Key* checked, then row ID values are generated by the object manager. Otherwise, a user-entered row ID value is assumed, and the object manager does not generate a row ID value for it.

  However, in either configuration, the Primary Key column must not use the Oracle Sequence Object.
- All EBCs require the Siebel S_APP_VER and S_SYS_PREF tables to be present in the external database.

  Siebel CRM uses the UNICD_DATATYPS_FLG column of the S_APP_VER table to indicate whether the database is a Unicode database. A value of 8 means UTF-8, and Y means UTF-16. For Non-Unicode databases, the Enterprise DB Server Code Page system preference is also required to have the correct setting.

  For help with creating and populating these tables, contact your Oracle sales representative for Oracle Advanced Customer Services to request assistance from Oracle's Application Expert Services.
- For EBCs that contain multivalue groups, if a primary join is enabled, then both the parent and the child business components must be from the same data source. Multivalue groups are also supported as long as such configuration does not require that a distributed join or a subquery be performed.
- Siebel visibility control (for example, ViewMode) is not supported for EBCs.
- An external join alias must have the same name as the name used for the external table.
- EBCs based on Database views can be used for queries only; updates are not supported.
- For EBCs that have a parent-child relationship, their related external tables must not have a foreign key constraint set between them on the external database. If they do have a foreign key constraint, then Copy and Deep Copy functionality will not work.

> **Note:** Significant configuration effort and changes might be required if you choose to reconfigure a standard Siebel business component on an external table. For example, existing join and link definitions are unlikely to function, because the source fields and underlying columns might not exist in the external table.

About Using External Business Components with the Siebel Web Clients

If EBCs are used with the Siebel Web or Mobile Web Clients, then new data sources corresponding to the data sources specified for the external tables must be added to the specific Siebel application configuration file. If the user name and password for the external data source are different from the current data source, then a log-in window appears to initiate logging into the external data source.

About Overriding Connection Pooling Parameters for the Data Source

Overriding the connection pooling parameters for the data source is supported. If connection pooling is enabled for the component but not for the data source, then set to zero (0) the following:

- DB Multiplex - Max Number of Shared DB Connections (DSMaxSharedDbConns)
- DB Multiplex - Min Number of Shared DB Connections (DSMinSharedDbConns)
- DB Multiplex - Min Number of Dedicated DB Connections (DSMinTrxDbConns) parameters for the data source

About Joins to Tables in External Data Sources

Joins from business components, based on the default data source to a table in an external data source, are supported in the Siebel application.

Like other joined fields, the fields based on the join to the EBC are read-only.

The limitations for joining business components to tables in an external data source are as follows:

- The source field for the join must be based on a table in the default data source.
- The destination column of the join must be the column mapped to System Field Id.
- Multiple single join specifications are not supported for the join to the external table.
- Reverse navigation (for example, a call to go to the last record) is not supported when fields from multiple data sources are active.

Join Constraints are supported. Joins to more than one external table might be specified. However, increasing the number of external joined data sources can cause degradation in performance.

Searching and Sorting on Fields Joined to External Tables

Fields based on a join to an external table can be searched and sorted. However, limitations do exist. The limitations for searching and sorting on fields joined to an external table follow:

- All fields in the sort specification must either be based on columns in the same external table, or be based on columns in the default data source.
- Named search specifications cannot be set on fields from an external data source.

For information on named search specifications, see the topic on the SetNamedSearch method in *Siebel Object Interfaces Reference* .

Performance tests are recommended if searching and sorting are permitted on fields based on joins to the external tables. The Siebel application does not have information on the data shape in the external tables. The Siebel application follows a rule-based approach to decide the order in which to query the external tables.

For example, consider the case where there are search and sort specifications on the fields in the Siebel Data Source but none on the fields from the external data source. The Siebel application decides to query the Siebel tables first. Only the

rows matching the query specification in the current workset are retrieved from the external data source. As more rows are retrieved from the tables in the Siebel Data Source, the rows from the external data source are also retrieved.

The rules become complex when Search and Sort Specifications are applied to multiple data sources. The rules followed are based on the following requirements:

1. Retrieving the first few rows quickly
2. Shipping the least amount of data between the Siebel and external data sources
3. Eliminating a sort step

Step 2 and Step 3 might produce competing results. In that case, Step 2 takes precedence.

If, as result of the search and sort specifications in effect, then the external table on which the Sort is based is not the driving table, the Siebel application raises an error if more than 1000 rows are retrieved. Refine the query specification in the event of this error.

Directives specified using the Business Component User property *External DataSource Field Priority On Search* to allow hinting of the order in which the tables in the data sources will be queried are supported. These directives can be applied based on a knowledge of the data shape in the Siebel and external tables.

For example, using the following property values:

Property	Value
External DataSource Field Priority On Search: FieldA	1
External DataSource Field Priority On Search: FieldB	2

A query on Field A is likely to be selective. If there is a search specification on Field A, then the table that field A is based on is considered the driving table.

A query on Field B is likely to be selective. If there is a search specification on Field B and none on Field A, then the table that field B is based on is considered the driving table.

About Distributed Joins

Just as join objects can be configured in Siebel Tools and represent a 1:1 relationship between tables resident within the Siebel data model, join objects can be configured to represent a 1:1 relationship with tables external to the Siebel database. A distributed join is a 1:1 relationship between tables that spans two relational data sources. This allows a single, logical record to span multiple data sources. In using distributed joins, the join fields are read-only, and the join specification can consist only of a single field. This federated field support provides the ability for the Object Manager to perform the cross-database join.

Distributed joins are configured the same as standard joins. The query is distributed when the Data Source child object of the table provides a *hint* to the Object Manager (OM) to federate the query.

Configuring Distributed Joins and Federated Fields

To configure distributed joins, you perform the following high-level tasks:

- Implement the external data source (similar to what was done for EBCs).
- The Datasource child object of the Table provides a *hint* to the object manager to federate the query.
- Create the Join.
- Add the fields to the business component.

To configure distributed joins and federated fields

1. Create the Join point to your external table.
2. Create the Join Specification.

   This is similar to what you do when creating a standard Siebel join.
3. Add Field to Business Component.

   Add the fields from the external table to the business component using the join specified.

Usage and Restrictions for Distributed Joins

The following usage guidelines and restrictions apply to distributed joins:

- The source field for the distributed join must be based on a table in the business component's data source.
- The destination column of the distributed join must be a column mapped to the Id System Field.
- Multiple join specifications are not supported for a distributed join. However, join constraints are supported.
- Inner join is not supported for a distributed join.
- Reverse navigation (for example, a call to go to the last record) is not supported when the fields from multiple data sources are active.
- All fields in the sort specification must be from the same data source.
- All fields in the named search specifications must be from the default data source.

Troubleshooting External Business Components

As you create EBCs, it is recommended that you consider the following steps:

1. Configure EBCs for *read* and make sure that the data is displayed correctly in the application.

   If the development team feels that some fields require script in order to display correctly then defer the implementation of these fields until testing is complete for a simple read.
2. Add any data transformation script or configuration required in order to provide read access to the more *complex* fields for display.
3. Configure EBCs for *update* and make sure that the data is stored correctly in the external database(s) and displayed correctly in the Siebel application.

Do not add any validation logic to the EBC at this time.

4. Once testing of data update is complete, establish any data transformation configuration or script required to update the fields.

   Make sure that the configuration uses script, which is preferred. However, it is recommended that any data transformation scripts be written on the *Pre* event.

   Data manipulation configuration and scripts must be attached to *Post* events.

As part of the troubleshooting process associated with EBCs, increasing the tracing level for a number of component events is suggested.

To increase the tracing level of component events

1. Navigate to the Administration - Server Configuration screen, Servers view, Components, Events, and then select the object manager being used.
2. Change the Log Level for the following Event Types to a higher value (the default is 1).

   Initially a value of 4 is recommended.

   - Task Configuration
   - DBC Log
   - SQL
   - Object Manager DB Connection Operation Log
   - General Object Manager Log
   - Object Manager Session Operation and SetErrorMsg Log
   - Object Manager runtime repository Operation and SetErrorMsg Log
   - Security Adapter Log

Following this change, restarting the affected components is recommended. With the increase log level, more information is stored in the relevant log files. Reset these values back to 1 when troubleshooting is completed.

11 Predefined EAI Business Services

Predefined EAI Business Services

This chapter lists the business services provided for Siebel EAI. Siebel CRM provides a number of business services. These services do not require any modification, but they do require that you choose and configure them to suit your requirements. It includes the following topic:

- *Predefined EAI Business Services*

For general information on using business services, see *Business Services*.

Predefined EAI Business Services

The following table presents the predefined Siebel EAI business services.

Business Service	Class	Description
EAI BTS COM Transport	CSSEAIBtsComService	EAI Siebel to BTS COM Transport.
EAI Data Transformation Engine	CSSDataTransformationEngine	EAI Data Transformation Engine (DTE). For information, see *Business Processes and Rules: Siebel Enterprise Application Integration* . The display name for this business service is EAI Data Mapping Engine.
EAI Dispatch Service	CSSEAIDispatchService	Dispatch Service. For information, see *Business Processes and Rules: Siebel Enterprise Application Integration* .
EAI DLL Transport	CSSDllTransService	EAI DLL Transport. For information, see *Transports and Interfaces: Siebel Enterprise Application Integration* .
EAI HTTP Transport	CSSHTTPTransService	EAI HTTP Outbound Transport. For information, see *Transports and Interfaces: Siebel Enterprise Application Integration* .
EAI Import Export	CSSEAIImportExportService	EAI Import Export Service. Imports integration objects from, or exports them to, XML files.

Business Service	Class	Description
		Note: This business service is intended only for Siebel user interface sessions in the Siebel Web Client or Siebel Mobile Web Client. For information on converting integration objects to and from XML files, see the chapter on Siebel XML converters in *XML Reference: Siebel Enterprise Application Integration* .
EAI Integration Object to XML Hierarchy Converter	CSSEAIIntObjHierCnvService	EAI Integration Object Hierarchy (also known as SiebelMessage) to XML hierarchy converter service. For information, see *XML Reference: Siebel Enterprise Application Integration* .
EAI MIME Doc Converter	CSSEAIMimeService	MIME Document Conversion Service. For information, see *Siebel EAI and File Attachments*.
EAI MIME Hierarchy Converter	CSSEAIMimePropSetService	EAI MIME Hierarchy Conversion Service. For information, see *Siebel EAI and File Attachments*.
EAI MQSeries Server Transport	CSSMqSrvTransService	EAI MQSeries Server Transport.
EAI MSMQ Transport	CSSMsmqTransService	EAI MSMQ Transport.
EAI Null Envelope Service	CSSEAINullEnvelopeService	EAI Null Envelope Service. For information, see *XML Reference: Siebel Enterprise Application Integration* .
EAI Query Spec Service	CSSEAIQuerySpecService	Used internally by the EAI Siebel Adapter to convert the SearchSpec method argument as a string to an Integration Object Instance that the EAI Siebel Adapter can use as a Query By Example object.
EAI Siebel Adapter	CSSEAISiebelAdapterService	EAI Siebel Adapter. For information, see *EAI Siebel Adapter Business Service*.
EAI Transaction Service	CSSBeginEndTransactionService	EAI Transaction service for working with Siebel transactions, such as begin and end, to find out whether in transaction.
EAI UI Data Adapter	CSSEAIUDAdapterService	EAI UI Data Adapter. For information, see *EAI UI Data Adapter Business Service*.
EAI XML Converter	CSSEAIXMLCnvService	Converts between XML and EAI messages. For information, see *XML Reference: Siebel Enterprise Application Integration* .
EAI XML Read from File	CSSEAIXMLPrtService	Reads an XML file and parses to a property set. For information, see *XML Reference: Siebel Enterprise Application Integration* .

Business Service	Class	Description
EAI XML Write to File	CSSEAIXMLPrtService	Prints a property set to a file as XML. For information, see *XML Reference: Siebel Enterprise Application Integration* .
EAI XML XSD Generator	CSSEAISchXSDService	Used to generate an XSD file from an integration object.
EAI XSD Wizard	CSSXMLSchemaWizard	Used to create integration objects based on XSD files.
EAI XSLT Service	CSSXSLTService	EAI XSL Transformation Service. Supports the Apache Xalan API as the XLST processor and Xerces as the XML parser.
Read CSV File	CSSCsvParserService	Converts a CSV file to a property set, and can then convert the property set to XML.
Siebel Message Envelope	CSSEAISMEnvelopeService	EAI Siebel Message Envelope Service. For information, see *XML Reference: Siebel Enterprise Application Integration* .
XML Converter	CSSXMLCnvService	Converts between XML documents and arbitrary Property Sets. For information, see *XML Reference: Siebel Enterprise Application Integration* .
XML Hierarchy Converter	CSSXMLCnvService	Converts between XML documents and XML Property Set or Arbitrary Property Set. For information, see *XML Reference: Siebel Enterprise Application Integration* .

12 Property Set Representation of Integration Objects

Property Set Representation of Integration Objects

This chapter describes the relationship between property sets and integration objects. Property sets are in-memory representations of integration objects. For an overview of property sets, see *Using Siebel Tools* . It includes the following topics:

- *Property Sets and Integration Objects*
- *Example Instance of an Account Integration Object*

Property Sets and Integration Objects

Many EAI business services operate on integration object instances. Because business services take property sets as inputs and outputs, it is necessary to represent integration objects as property sets. The mapping of integration objects, components, and fields to property sets is known as the Integration Object Hierarchy.

Using this representation, you can pass a set of integration object instances of a specified type to an EAI business service. You pass the integration object instances as a child property set of the business service method arguments. This property set always has a type of SiebelMessage. You can pass the SiebelMessage property set from one business service to another in a workflow without knowing the internal representation of the integration objects.

Property Set Node Types

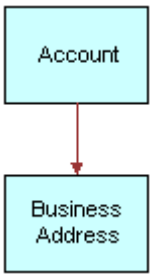
When passing integration object instances as the input or output of a business service, you can use property sets to represent different node types, as presented in the following table.

Name	Parent	Value of Type Attribute	Properties	Description
Service Method Arguments	Not applicable	Ignored	The properties of this property set contain the service specific parameters, such as *PrimaryRowId* for the EAI Siebel Adapter.	This is the top-level (highest-level) property set of a business service's input or output. The properties of this property set contain the service-specific parameters (for example, *PrimaryRowId* for the EAI Siebel Adapter).
SiebelMessage	Service Method Arguments	SiebelMessage	The properties of this property set contain header attributes associated with the	This property set is a wrapper around a set of integration object instances of a specified

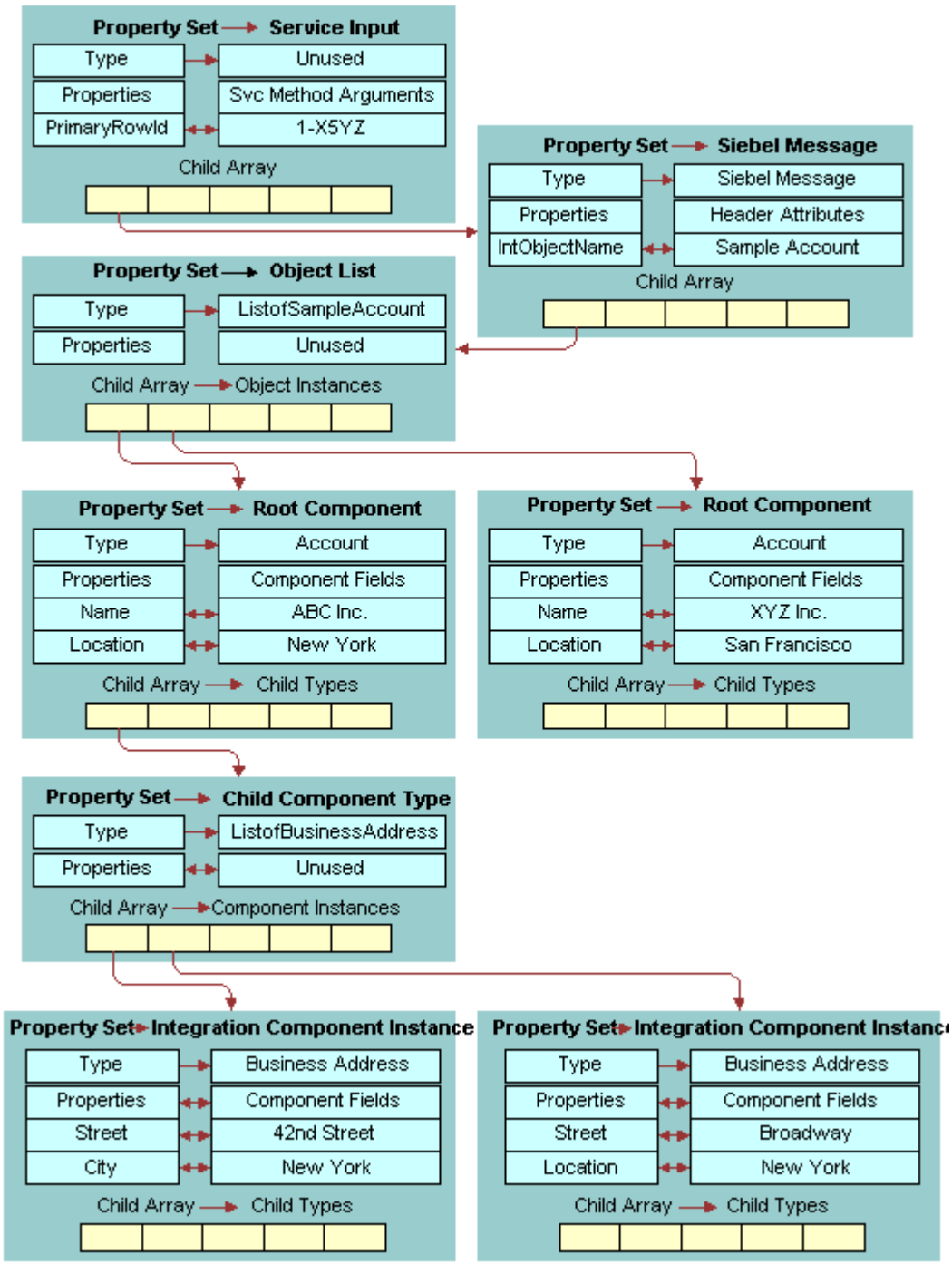
Name	Parent	Value of Type Attribute	Properties	Description
			integration object, for example, *IntObjectName*.	type. To pass integration objects between two business services in a workflow, this property set is copied to and from a workflow process property of type *Hierarchy*.
Object List	SiebelMessage	List*OfObjectType*	Not used.	This property set identifies the object type that is being represented. The root components of the object instances are children of this property set.
Root Component	Object List	*Root Component Name*	The property names of the property set represent the field names of the component, and the property values are the field values.	This property set represents the root component of an integration object instance.
Child Component Type	Root Component or Component	List*Of**Component Name*	Not used.	An integration component can have a number of child component types, each of which can have zero or more instances. The Integration Object Hierarchy format groups the child components of a given type under a single property set. This means that child components are actually grandchildren of their parent component's property set.
Child Components	Child Component Type	*Component Name*	The property names of the property set represent the field names of the component, and the property values are the field values.	This property set represents a component instance. It is a grandchild of the parent component's property set.

Example Instance of an Account Integration Object

This example shows an Account integration object in which the object has two component types: Account and Business Address (which is a child of Account). The hierarchy of component types, from the perspective of Oracle's Siebel Tools, looks like that shown in the following figure.



The following figure shows an example instance of this object type, using the Integration Object Hierarchy representation. There are two Sample Account instances. The first object instance has an Account component and two Business Address child components. The second object instance has only an Account component with no child components.


Property Set → Service Input
Type
Unused
Properties
Svc Method Arguments
PrimaryRowId
1-X5YZ
Child Array
Property Set → Siebel Message
Type
Siebel Message
Properties
Header Attributes
IntObjectName
Sample Account
Child Array
Property Set → Object List
Type
ListofSampleAccount
Properties
Unused
Child Array → Object Instances
Property Set → Root Component
Type
Account
Properties
Component Fields
Name
ABC Inc.
Location
New York
Child Array → Child Types
Property Set → Root Component
Type
Account
Properties
Component Fields
Name
XYZ Inc.
Location
San Francisco
Child Array → Child Types
Property Set → Child Component Type
Type
ListofBusinessAddress
Properties
Unused
Child Array → Component Instances
Property Set → Integration Component Instance
Type
Business Address
Properties
Component Fields
Street
42nd Street
City
New York
Child Array → Child Types
Property Set → Integration Component Instance
Type
Business Address
Properties
Component Fields
Street
Broadway
Location
New York
Child Array → Child Types

13 DTDs for XML Gateway Business Service

DTDs for XML Gateway Business Service

This chapter lists the various inbound and outbound Document Type Definitions (DTDs) for the XML Gateway business service. It covers the following topics:

- *Outbound DTDs for the XML Gateway Business Service*
- *Inbound DTDs for the XML Gateway Business Service*

Outbound DTDs for the XML Gateway Business Service

The following sections contain examples of DTDs representing the %methodName% request sent from the XML Gateway to the external application.

Delete

The following DTD is for the Delete request:

```
<!ELEMENT siebel-xmlext-delete-req (buscomp, remote-source, row)>
<!ELEMENT buscomp (#PCDATA)>
<!ATTLIST buscomp id NMTOKEN #REQUIRED>
<!ELEMENT remote-source ( #PCDATA )*>
<!ELEMENT row (value+)>
<!ELEMENT value (#PCDATA)*>
<!ATTLIST value field CDATA #REQUIRED>
```

Init

The following DTD is for the Init request:

```
<!ELEMENT siebel-xmlext-fields-req (buscomp, remote-source?)>
<!ELEMENT buscomp (#PCDATA)>
<!ATTLIST buscomp id NMTOKEN #REQUIRED >
<!ELEMENT remote-source (#PCDATA)*>
```

Insert

The following DTD is for the Insert request:

```
<!ELEMENT siebel-xmlext-insert-req (buscomp, remote-source?, row)>
<!ELEMENT buscomp (#PCDATA)>
<!ATTLIST buscomp id NMTOKEN #REQUIRED>
<!ELEMENT remote-source (#PCDATA)*>
<!ELEMENT row (value+)>
```

```
<!ELEMENT value (#PCDATA)*>
<!ATTLIST value field CDATA #REQUIRED>
```

PreInsert

The following DTD is for the PreInsert request:

```
<!ELEMENT siebel-xmlext-preinsert-req (buscomp, remote-source?)>
<!ELEMENT buscomp (#PCDATA)>
<!ATTLIST buscomp id NMTOKEN #REQUIRED >
<!ELEMENT remote-source (#PCDATA)*>
```

Query

The following DTD is for the Query request:

```
<!ELEMENT siebel-xmlext-query-req (buscomp , remote-source?, max-rows?, search-
string?, match?, search-spec?, sort-spec? )>
<!ELEMENT buscomp (#PCDATA)>
<!ATTLIST buscomp id NMTOKEN #REQUIRED>
<!ELEMENT remote-source (#PCDATA)*>
<!ELEMENT max-rows (#PCDATA)>
<!ELEMENT search-string (#PCDATA)>
<!ELEMENT match (#PCDATA)>
<!ATTLIST match field CDATA #REQUIRED>
<!ELEMENT search-spec (node)>
<!ELEMENT node (#PCDATA | node)*>
<!ATTLIST node node-type (Constant | Identifier | Unary Operator | Binary Operator)
#REQUIRED>
<!ATTLIST node value-type (TEXT | NUMBER | DATETIME | UTCDATETIME | DATE | TIME)
#IMPLIED>
<!ELEMENT sort-spec (sort+)>
<!ELEMENT sort (#PCDATA)>
<!ATTLIST sort field CDATA #REQUIRED>
```

Update

The following DTD is for the Update request:

```
<!ELEMENT siebel-xmlext-update-req (buscomp, remote-source?, row)>
<!ELEMENT buscomp (#PCDATA)>
<!ATTLIST buscomp id NMTOKEN #REQUIRED>
<!ELEMENT remote-source (#PCDATA)*>
<!ELEMENT row (value+)>
<!ELEMENT value (#PCDATA)*>
<!ATTLIST value changed ( true | false ) #REQUIRED>
<!ATTLIST value field CDATA #REQUIRED>
```

Inbound DTDs for the XML Gateway Business Service

The following sections contain examples of DTDs representing the %methodName% response sent from the external application to the XML Gateway.

Delete Response

The following DTD is for the Delete response:

```
<!ELEMENT siebel-xmlext-dekete-ret EMPTY >
```

Init Response

The following DTD is for the Init response:

```
<!ELEMENT siebel-xmlext-fields-ret (support+)>
<!ELEMENT support EMPTY >
<!ATTLIST support field CDATA #REQUIRED>
```

Insert Response

The following DTD is for the Insert response:

```
<!ELEMENT siebel-xmlext-preinsert-ret (row)>
<!ELEMENT row (value+)>
<!ELEMENT value (#PCDATA)*>
<!ATTLIST value field CDATA #REQUIRED >
```

PreInsert Response

The following DTD is for the PreInsert response:

```
<!ELEMENT siebel-xmlext-preinsert-ret (row)>
<!ELEMENT row (value)*>
<!ELEMENT value (#PCDATA)*>
<!ATTLIST value field CDATA #REQUIRED >
```

Query Response

The following DTD is for the Query response:

```
<!ELEMENT siebel-xmlext-query-ret (row*)>
<!ELEMENT row (value+)>
<!ELEMENT value (#PCDATA)*>
```

```
<!ATTLIST value field CDATA #REQUIRED >
```

Update Response

The following DTD is for the Update response:

```
<!ELEMENT siebel-xmlext-update-ret (row)>
<!ELEMENT row (value+)>
<!ELEMENT value (#PCDATA)>
<!ATTLIST value field CDATA #REQUIRED >
```