



INTEGRATION PLATFORM TECHNOLOGIES: SIEBEL eBUSINESS APPLICATION INTEGRATION VOLUME II

VERSION 7.5.3

AUGUST 2003

12-FRX505

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

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

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

Other product names, designations, logos, and symbols may be trademarks or registered trademarks of their respective owners.

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

Proprietary Information

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

Contents

Introduction

How This Guide Is Organized	11
Additional Resources	12
Revision History	13

Chapter 1. About Integration Objects

Integration Objects Terminology	15
Siebel Integration Objects	16
Integration Object Base Object Type	18
Integration Object and Integration Object Instance	19
Siebel Integration Object Wizards	21
Structure of Siebel Integration Objects	22
Associations	26
Multi-Value Groups	27
Picklists	33
Calculated Fields	35
Inner Joins	35
Operation Control	36
Field Dependencies	37
Primaries	37
Repository Objects	37
Integration Component Keys	38
User Keys	38

Status Keys	44
Hierarchy Parent Key	44
Hierarchy Root Key	45

Chapter 2. Creating and Maintaining Integration Objects

Integration Object Builder Overview	47
Creating Integration Objects Using the EAI Siebel Wizard	48
Siebel Integration Object Fine-Tuning	51
Integration Object Validation	51
Integration Objects Synchronization	52
Synchronization Considerations	52
Synchronization Rules	57
The EAI Siebel Wizard	62
Siebel Integration Objects Maintenance and Upgrade	64
Permission Rules for Integration Components	65
EAI Siebel Adapter Access Control	66
Integration Object User Properties	66
Example of an Integration Object With M:M Relationship	70
Generating Schemas	72
Performance Considerations	73
Business Component Restrictions	74
Best Practices	74

Chapter 3. Business Services

Overview of Business Services	77
Creating Business Services	78
Business Service Structure	79
About Property Sets	80
Creating Business Services in Siebel Tools	82

Defining a Business Service in Siebel Tools	83
Defining Business Service Methods	84
Defining Business Service Method Arguments	84
Defining and Writing Business Service Scripts	85
Specifying Business Service Subsystems	86
Defining Business Service User Properties	87
Creating a Business Service in the Siebel Client	87
Business Service Export and Import	88
Testing Your Business Service	89
Accessing a Business Service Using Siebel eScript or Siebel VB	90
Business Scenario	91
Code Sample	93

Chapter 4. Web Services

Web Services Overview	95
Inbound Web Services	97
Outbound Web Services	100
XML Schema Support for <xsd:any> Tag	105
Examples of Invoking Web Services	107
Troubleshooting Tips	115

Chapter 5. EAI Siebel Adapter

EAI Siebel Adapter Overview	117
EAI Siebel Adapter Methods	117
Query Method	122
QueryPage Method	123
Synchronize Method	123
Upsert Method	124
Insert Method	125
Update Method	125

Delete Method	126
Execute Method	126
XML Examples	129
MVGs in EAI Siebel Adapter	131
Search Specification	134
Language-Independent Code	138
EAI Siebel Adapter Concurrency Control	139
Modification Key	140
Modification IDs	140
Siebel eAI and Run-Time Events	145

Chapter 6. Siebel eAI and File Attachments

Exchange of Attachments with External Applications	147
Using MIME Messages to Exchange Attachments	148
Creating the Integration Object	149
Creating Workflow Processes Examples	150
The EAI MIME Hierarchy Converter	156
Outbound Integration	157
Inbound Integration	158
The EAI MIME Doc Converter	159
EAI MIME Doc Converter Properties	160

Chapter 7. Siebel Virtual Business Components

Overview of Virtual Business Components	163
Enhancements to VBCs for This Version	165
Usage and Restrictions	165
Virtual Business Components	166
Creating a New Virtual Business Component	167
Setting User Properties for the Virtual Business Component	168
XML Gateway Service	169

XML Gateway Methods	172
XML Gateway Method Arguments	173
Examples of Outgoing XML Format	174
Search-Spec Node-Type Types	179
Examples of Incoming XML Format	180
External Application Setup	183
Custom Business Service Methods	183
Common Method Parameters	184
Business Services Methods and Their Property Sets	185
Custom Business Service Example	203

Appendix A. Predefined EAI Business Services

Predefined EAI Business Services	219
--	-----

Appendix B. Property Set Representation of Integration Objects

Property Sets and Integration Objects	223
Property Set Node Types	224
Example of a Sample Account	226

Appendix C. DTDs for XML Gateway Business Service

Outbound DTDs	229
Inbound DTDs	231

Index

Introduction

This guide explains the details of Siebel eBusiness Application Integration's (Siebel eAI's) integration platform technologies, including integration objects, business services, EAI Siebel Adapter, virtual business components, and so on.

The audience for this guide consists primarily of employees in these categories:

Business Analysts	Persons responsible for analyzing application integration challenges and planning integration solutions at an enterprise.
Database Administrators	Persons who administer the database system, including data loading, system monitoring, backup and recovery, space allocation and sizing, and user account management.
Siebel Application Administrators	Persons responsible for planning, setting up, and maintaining Siebel applications.
Siebel Application Developers	Persons who plan, implement, and configure Siebel applications, possibly adding new functionality.
Siebel Integration Developers	Persons responsible for analyzing a business situation or using the analysis of a business analyst to build the integration solution for Siebel applications at an enterprise.
Siebel System Administrators	Persons responsible for the whole system, including installing, maintaining, and upgrading Siebel applications.
System Integrators	Persons responsible for analyzing a business situation or using an analysis to build integration solutions or to develop custom solutions for specific applications at an enterprise.

The audience for this book also needs to have experience in data integration, data transformation (data mapping), scripting or programming, and XML.

Product Modules and Options

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

How This Guide Is Organized

This book is organized in a way that presents the most important information up front. It describes how to create and maintain integration objects and then provides description of individual components of the EAI integration platform, such as virtual business components.

This book is Volume 2 of a five-volume set. The full set includes:

- *Overview: Siebel eBusiness Application Integration Volume I*
- *Integration Platform Technologies: Siebel eBusiness Application Integration Volume II*
- *Transports and Interfaces: Siebel eBusiness Application Integration Volume III*
- *Business Processes and Rules: Siebel eBusiness Application Integration Volume IV*
- *XML Reference: Siebel eBusiness Application Integration Volume V*

Additional Resources

The product documentation set for Siebel eBusiness Applications is provided on the *Siebel Bookshelf CD-ROM* or in *Siebel Online Help*. The following integration related books and online help describe all the tools required to implement integration:

- *Siebel Tools Online Help*.
- *Siebel Tools Reference*.
- *Siebel Business Process Designer Administration Guide*.
- *Siebel Enterprise Integration Manager Administration Guide* if you perform bulk loading or unloading of data.
- *Application Services Interface Reference*.

The Connector books provide specifics on each of the associated connectors.

Revision History

eAI Volume II: Integration Platform Technologies

Version 7.5.3

Table 1. Changes Made in Version 7.5.3

Topics	Revision
"XML Schema Support for <xsd:any > Tag"	New for 7.5.3: Support of the <xsd:any > tag added to Chapter 4, "Web Services."
"Web Services Support for Transport Headers"	New for 7.5.3: Support for Transport Headers in Chapter 4, "Web Services."

Version 7.5, Rev. A

Table 2. Changes Made in Version 7.5, Rev. A

Topics	Revision
"Integration Object Base Object Type"	Added a new table presenting Integration Object Types. Chapter 1, "About Integration Objects."
"Picklists"	Changed PicklistUserKey to PicklistUserKeys and added requirements in Chapter 1, "About Integration Objects."
"Picklists"	Added this user property to Table 6 in Chapter 2, "Creating and Maintaining Integration Objects."
"Example of an Integration Object With M:M Relationship"	Added a new section discussing integration objects with M:M in Chapter 2, "Creating and Maintaining Integration Objects."
"Inner Joins"	Added new content in Chapter 2, "Creating and Maintaining Integration Objects."
"Best Practices"	Added new scenarios in Chapter 2, "Creating and Maintaining Integration Objects."

Table 2. Changes Made in Version 7.5, Rev. A

Topics	Revision
"Code Sample"	Changed CanInvoke == "TRUE" to CanInvoke = "TRUE" in Chapter 3, "Business Services."
"Accessing a Business Service Using Siebel eScript or Siebel VB"	New examples on calling business services in Chapter 3, "Business Services."
"Web Services Overview"	Provided more details throughout the Chapter 4, "Web Services" .
Chapter 7, "Siebel Virtual Business Components"	Updated all the examples in Chapter 7, "Siebel Virtual Business Components."
"Insert Method"	Updated the definition for Update method in Chapter 5, "EAI Siebel Adapter."
"Query Method"	Added a new example for Query method in Chapter 5, "EAI Siebel Adapter."
"Custom Business Service Example"	Added a new example in Chapter 7, "Siebel Virtual Business Components."

About Integration Objects1

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.

Integration Objects Terminology

This chapter describes the concepts that are often referred to using different terminology from one system to another. [Table 3](#) has been included to clarify the information in this chapter by providing a standard terminology for these concepts.

Table 3. Terminology

Term	Description
Metadata	Data that describes data. For example, the term datatype describes data elements such as char, int, Boolean, time, date, and float.
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.
Component	A constituent part of any generic object.
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.
Field	A generic reference to a data structure that can contain one data element.

Table 3. Terminology

Term	Description
Siebel integration component field	A data structure that can contain one data element in a Siebel integration component.
Siebel integration component	A constituent part of a Siebel 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.
Siebel integration object	An object stored in the Siebel repository that represents some Siebel business object.
Integration message	A bundle of data consisting of two major parts: header information that describes what should be done with or to the message itself, and instances of integration objects, that is, data in the structure of the integration object.

Siebel Integration Objects

Siebel integration objects allow you to represent integration metadata for Siebel business objects, XML, SAP IDOCs, and SAP BAPIs as common structures that the 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, see *Business Processes and Rules: Siebel eBusiness Application Integration Volume IV*.

The typical integration project involves transporting data from one application to another. For example, you may want to synchronize data from a back-office system with the data in your Siebel application. You may 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, you should 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 should not be replicated in the receiving data source, or that should 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.

Integration Object Base Object Type

Each integration object created in Siebel Tools has to be based on one of the base object types presented in [Table 4](#). 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.

Table 4. Base Object Types

Name	Description
None	For internal use only.
OLE DB	Used to expose Siebel business component as OLEDB rowset that can be used by OLEDB consumers such as Excel, Word, and so on. The OLE DB Provider only accepts integration objects of this type.
SAP BAPI Input	Used to represent the input structure of an SAP RFC or BAPI function call. For details, see <i>Siebel eBusiness Connector for SAP R/3 Guide</i> .
SAP BAPI Output	Used to represent the output structure of an SAP RFC or BAPI function call. For details, see <i>Siebel eBusiness Connector for SAP R/3 Guide</i> .
SAP IDOC	Used with the IDOC Adapter and Receiver in version 6.x and 7.0. For details, see <i>Siebel eBusiness Connector for SAP R/3 Guide</i> .
SAP IDOC Adapter	Used to represent an SAP IDOC structure. For details, see <i>Siebel eBusiness Connector for SAP R/3 Guide</i> .
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. EAI Siebel Adapter only accepts integration object of this type.

Table 4. Base Object Types

Name	Description
Table	Obsolete.
XML	Used to represent external XML Schema such as DTD or XSD. For details on DTD and XSD, see <i>XML Reference: Siebel eBusiness Application Integration Volume V</i> .

Integration Object and Integration Object Instance

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 thing.

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. [Figure 1](#) illustrates a simple example of an integration object and an integration object instance, using partial data.

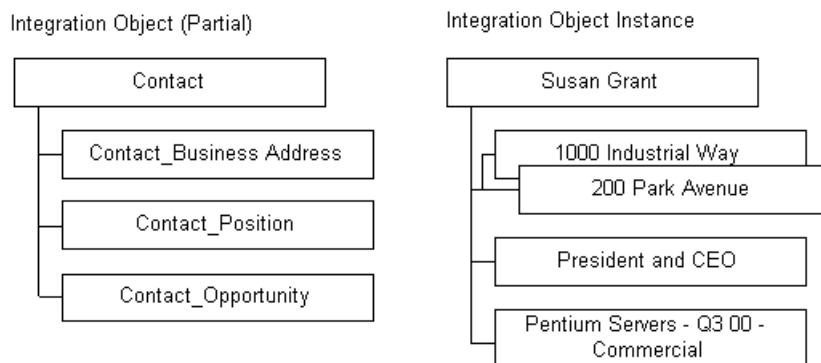


Figure 1. Integration Object and Integration Object Instance

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

Siebel 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, and others that create integration objects for external systems based on Siebel objects. [Figure 2](#) shows the logic of Integration object Wizard and Generate Schema Wizard. The Generate Code wizard (not shown) works in the same manner as the Generate Schema wizard, but it generates Java classes.

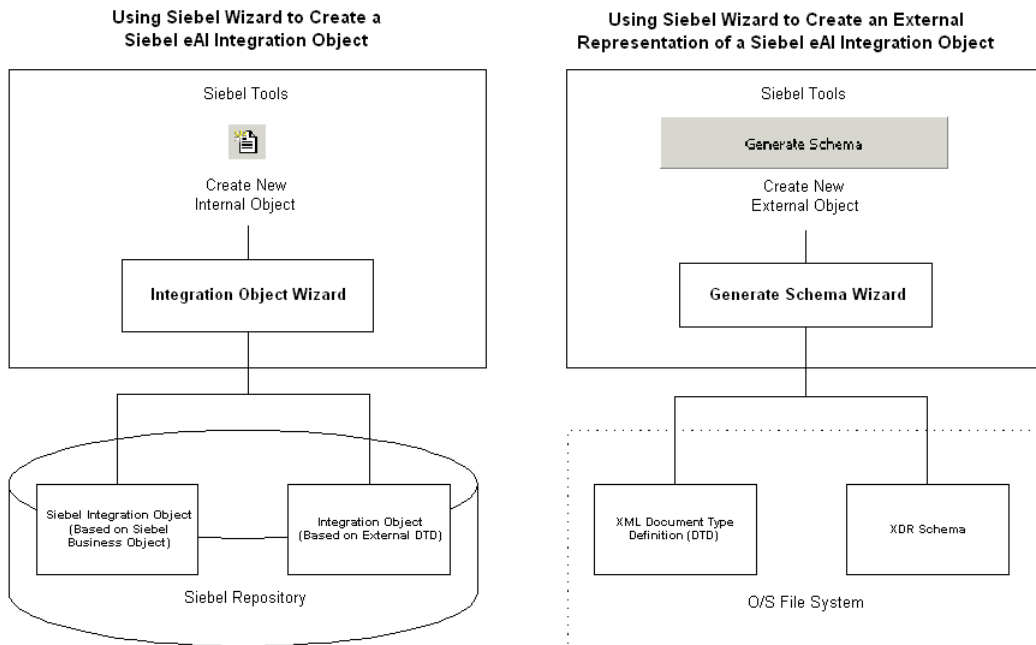


Figure 2. Integration Object Wizards

- **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 representations of external business objects using XML Schema Definition (XSD) or Document Type Definition (DTD). To access this wizard, navigate to the New Object dialog box in Siebel Tools and after selecting the EAI tab, double-click the Integration Object icon 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. To access this wizard, navigate to the Integration Objects list in Siebel Tools and 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. To access this wizard, navigate to the Integration Objects list in Siebel Tools object explorer and 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 eBusiness Application Integration documentation set where appropriate.

Structure of Siebel Integration Objects

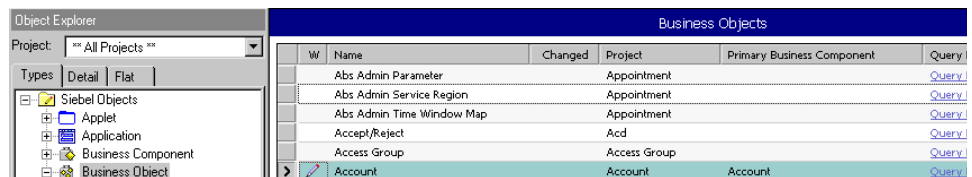
The Siebel integration object provides a hierarchical structure that represents a complex data type. Most specifically, prebuilt eAI integration objects describe the structure of Siebel business objects, SAP IDOCs, SAP BAPIs, XML, and external data. Most integration projects require the use of an integration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

Chapter 2, “Creating and Maintaining Integration Objects” continues with descriptions of 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: You should 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.

Siebel business objects conform to a particular structure in memory. Although it is generally not necessary to consider this structure when working with Siebel applications, 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. Figure 3 shows the Account business object in Siebel Tools.



W	Name	Changed	Project	Primary Business Component	Query
	Abs Admin Parameter		Appointment		Query
	Abs Admin Service Region		Appointment		Query
	Abs Admin Time Window Map		Appointment		Query
	Accept/Reject		Acid		Query
	Access Group		Access Group		Query
	Account		Account	Account	Query

Figure 3. Account Parent Business Component

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, you will see that each component can have one or more fields. [Figure 4 on page 25](#) 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.

Figure 4 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 may 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.

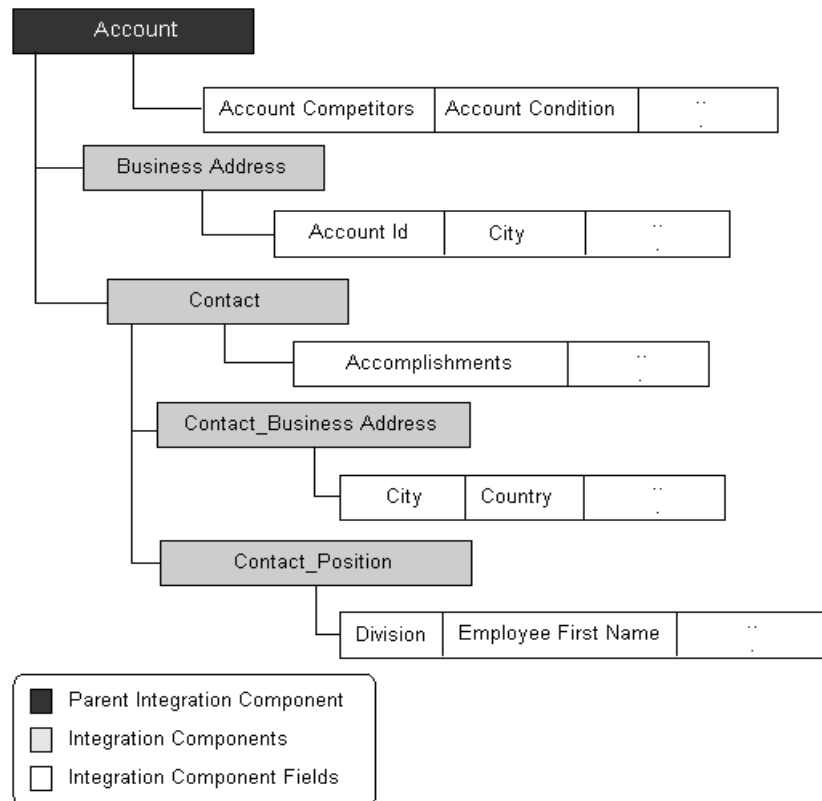


Figure 4. Representation of Partial Account Integration Object

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 multi-value 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 [Figure 5](#), the Opportunity integration component has one user property defined: the *Association* user property, set to a value of Y.

Integration Components					
	W	External Name Context	Name	Changed	Parent Integration Component
		Contact	Contact	✓	Contact
		Opportunity	Opportunity	✓	Contact
Integration Component User Props					
	W	Name	Changed	Value	Inactive
		Association	✓	Y	

Figure 5. Integration Component Representation of Association

NOTE: When building an integration object, if an integration component is an association based on an intersection table, the user key for this integration component cannot contain fields based directly or indirectly on the same association intersection table.

Multi-Value Groups

Multi-value groups (MVGs) are used within Siebel business components to represent database multivalued 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 may 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 should 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, the MVG is represented in several screens as illustrated in the following sections.

Screen 1: Fields View

For example, as illustrated in Figure 6, the Account business component contains a multi-value group field, the Address Id.

Business Components					
	W	Name	Changed	Project	Cache Data Class
		Access Control Party Reporting		System	CSSBusComp
		Access Control Test		System	CSSBusComp
		Access Group		Access Group	CSSBCGroup
		Access Group Member		Access Group	CSSBCBase
>		Account		Account	CSSBCBase
<					
Fields					
	Required	W	Name	Changed	Dest Field Multi Value Link
			Address Active Status		Active Status Business Address
>			Address Id		Id Business Address

Figure 6. Address Id MVG Field in the Account Business Component

Screen 2: Multi-Value Links

As shown in [Figure 7](#), the multi-value link property has the value Business Address. If you navigate to the Multi Value Link screen, you see that the Business Address multi-value link has the value Business Address as its Destination Business Component.

Business Components									
W	Name	Changed	Project	Cache Data	Class	Data Source	Dirty Reads	Distinct	
	Access Control Party Reporting Relation		System		CSSBusComp		✓		
	Access Control Test		System		CSSBusComp		✓		
	Access Group		Access Group		CSSBCGroup		✓		
	Access Group Member		Access Group		CSSBCBase		✓		
➤	Account		Account		CSSBCBase		✓		
⏪									
Multi Value Links									
W	Name	Auto Primary	Primary Id Field	Destination Business Component	Destination Link				
	Account Category	Default	Primary Category Id	Account Category	Account/Account Category				
	Account Credit Profile	Default		Account Credit Profile	Account/Account Credit Profile				
	Account Synonym	Default	Primary Synonym Id	Account Synonym	Account/Account Synonym				
	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				

Figure 7. Destination Business Component

Screen 3: Fields View

As shown in [Figure 8](#), the Business Address multi-value link has Business Address as its Destination Business Component. This means that there is another business component named Business Address that contains the fields that are collectively represented by Address Id in the Account business component.

Business Components

	W	Name	Project	Table
>		Business Address	Contact	S_ADDR_ORG
		Business Service	Business Service	S_RT_SVC
		Business Service Input Argument Properties	Business Service	
		Business Service Method	Business Service	S_RT_SVC METH
		Business Service Method Arg	Business Service	S_RT_SVC M_ARG
<				

Fields

Required	W	Name	Changed	Column	Dest Field
		Account Id		OU_ID	
		Active Status		ACTIVE_FLG	
>		Address Name		ADDR_NAME	
		Address Name Locked		NAME_LOCK_FLG	

Figure 8. Business Address Business Component

Graphical Representation

Figure 9 shows a graphical way to represent the relationship between Account business component and the Business Address multi-value link.

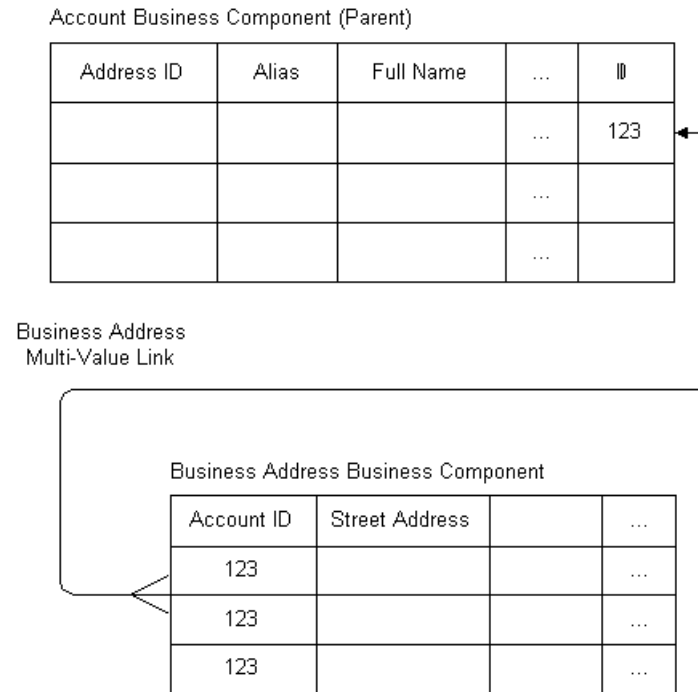


Figure 9. Address Id Field and Business Address MVG

The more table-like representation above shows how the Business Address multi-value 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 an Integration Component

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 Multi-value Link property and the Multi-value 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, 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*.

Figure 10 shows an integration component based on an MVG and its user property value in Siebel Tools.






Integration Components			
W	External Name Context	Name	Parent Integration
	Account_Bill To Contact	Account_Bill To Contact	Account
>	Account_Business Address	Account_Business Address	Account
	Account_Industry	Account_Industry	Account
	Account_Organization	Account_Organization	Account
	Account_Organization Unit Type	Account_Organization Unit Type	Account

Integration Component User Props			
W	Name	Changed	Value
>	MVG	✓	Y

Figure 10. Integration Component Based on MVG Business Component

The EAI Siebel Adapter needs to know the names of the MVG fields as they are defined in the parent business component—in this example, Account—and also the names of the MVG fields as they are known in the business component that represents the MVG—in this example, Account Business Address. As shown in Figure 11, the integration component fields represent the MVG.

Integration Components

	W	External Name Context	Name	Parent Integration
		Account_Bill To Contact	Account_Bill To Contact	Account
>		Account_Business Address	Account_Business Address	Account
		Account_Industry	Account_Industry	Account
		Account_Organization	Account_Organization	Account
		Account_Organization Unit Type	Account_Organization Unit Type	Account
<				

Integration Component Fields






	Inactive	W	Name	Changed	Data Type	Length
>			Address Active Status	✓	DTYPE_TEXT	
			Address Id	✓	DTYPE_ID	30
			Address Integration Id	✓	DTYPE_TEXT	30
			Address Name	✓	DTYPE_TEXT	100
			Bill Address Flag	✓	DTYPE_TEXT	

Figure 11. Integration Component Fields Representing MVG

To represent both names, each field is assigned an integration component field user property that contains the entry *MVGFieldName* or *AssocFieldName* if the user property is *MVGAssoc*. Its value is the name of the field shown in the parent business component—in this example, *Business Address*.

Picklists

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

If validation is done by EAI Siebel Adapter, and the pickmap for the picklist contains more than one field, when designing the integration object, you need to decide which of the fields to use as a search criterion and which to simply update if input values are different than those in the picklist (provided that picklist allows updates).

An example would be 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 needs to perform is to determine which of these fields should be 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 example above: 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 EAI Siebel Adapter to find the matching record in the picklist. (In the example above, this would be Account and Account Location.)

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

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').

In order for 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. Please 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 should 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 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 should also be present and empty in the input property set.

NOTE: Picklist validation in EAI Siebel Adapter is required for dynamic picklists. For details, see [“Picklist Validation” on page 73](#).

Calculated Fields

Calculated fields are inactive in the integration object when they are created. If your business needs require it, you need to activate the calculated fields in the integration object.

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

Inner Joins

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, you need to activate them.

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

For example, assume that Account business component has an inner join to 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 Account business component, not every account in Siebel Database is associated with a project. So, having Project Name active in the integration component would limit 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 project, then you can change the join to S_PROJ in the Account business component to an outer join. For details on join, see *Siebel Tools Reference*.

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

Operation Control

Each integration component has user properties that indicate if an Insert, Update, or Delete can be performed on the corresponding business component, indicated by a NoInsert, NoUpdate, or NoDelete. A similar user property NoUpdate may be set on an integration component field. If any of these user properties are set to Y, the corresponding business component method is used to validate the operation.

The user properties NoQuery and NoSynchronize are defined on integration components to specify to the Siebel Adapter if a corresponding operation is to be performed on an instance of that type. These properties take values Y or N.

The user property AdminMode set to Y indicates that the update of the corresponding business component is to be performed in admin mode. This can be defined on either integration object or integration component definitions.

The user properties IgnorePermissionErrorsOnUpdate, IgnorePermissionErrorsOnInsert, and IgnorePermissionErrorsOnDelete can be used to suppress the errors that arise from having the NoUpdate, NoInsert, and NoDelete user properties set to Y. The error is ignored and processing will continue when properties IgnorePermissionErrorsOnUpdate, IgnorePermissionErrorsOnInsert and IgnorePermissionErrorsOnDelete are set to Y.

Field Dependencies

Dependency between fields can be defined by user properties of the integration component field. The names of these user properties must start with FieldDependency, and the value of each property should contain the name of the field that the associated field is dependent on. The Siebel Adapter processes fields in the order defined by these dependencies and errors out if cyclic dependencies exist.

Siebel Adapter automatically takes into account dependencies of 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 the field C, then this implies dependencies of both A and B on C. As a consequence, Siebel Adapter will set field C before fields A and B.

Primaries

Primaries are set through multi-value links. However, you should not use multi-value links for modifying the linked component. To modify the linked component you should use links. If you need to set primaries in addition to modifying the linked component, use both links and multi-value links in your integration object. EAI Siebel Adapter should use the multi-value link only after it processes the component through the link; therefore, the link or the Association component should have a smaller external sequence number than the related MVG or MVGAssociation component. See [“Structure of Siebel Integration Objects” on page 22](#) for an example.

Repository Objects

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

Integration Component Keys

There are multiple types of integration component keys.

- **User Key.** See [“User Keys” on page 38](#).
- **Status Key.** See [“Status Keys” on page 44](#).
- **Hierarchy Parent Key.** See [“Hierarchy Parent Key” on page 44](#).
- **Hierarchy Root Key.** See [“Hierarchy Root Key” on page 45](#).
- **Modification Key.** See [“EAI Siebel Adapter Concurrency Control” on page 139](#).
- **Foreign Key.** See *Siebel eBusiness Connector for Oracle Guide*.
- **Target Key.** See *Siebel eBusiness Connector for Oracle Guide*.

NOTE: There should be just 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, EAI Siebel Adapter picks the first one and ignores the second one.

User Keys

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” on page 40](#). You should make sure that the generated user keys match your business requirements; otherwise, inactivate them or add new user keys as appropriate.

Integration component keys are built by the Integration Object Builder wizard based on values in the underlying table of the business component that the integration component is based on. 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.

In Siebel Tools, user keys are defined as integration component key objects, with Key Type property set to User Key.

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, 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 Account Name and Account Integration Id field present. When EAI Siebel Adapter performs validation, it first checks the Account and Account Location field (the first user key for the Account integration component). In this example, because the Account Location field is missing, 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 EAI Siebel Adapter uses that as the user key to match the record. Now if the same instance also had Account Location field present, but set to null, then EAI Siebel Adapter would have picked 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 of 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: EAI Siebel Adapter uses user keys to match integration component instances with business component records. Since the match is case sensitive there is a chance that records are not matched if the case of the user key fields do not match. To avoid this, use the Force Case property on the business component field to make sure that user key fields are always stored in one case.

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, determine the table on which your business component is based. In Siebel Tools, you can look up this information yourself. Navigate to the Business Components screen and select a business component and check the Table column.

You can then navigate to the Tables screen, locate the table you want—in this example, S_CONTACT—and open the User Keys applet to see the user keys defined for that table.

For example, as shown in [Figure 12](#), the table S_CONTACT has several user keys.

Tables							
		Extend	Apply	Activate			
W	Name	Changed	Project	User Name			
>	S_CONTACT		Newtable	Person			

Figure 12. User Keys for Table S_CONTACT

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 defined for a given user key must be exposed to the business component in which you are interested. For example, [Figure 13](#) shows three user key columns for the user key S_CONTACT_U1.

User Keys			
W	Name	Changed	User Key Type
	S_CONTACT: Scopus Migration		Scopus Migration
	S_CONTACT_EI		EI Index
	S_CONTACT_II		Integration Id
>	S_CONTACT_U1		Traditional U1 Index
	S_CONTACT_U1 - Std Replaced		Replaced

User Key Columns			
W	Name	Changed	Column
>	BU_ID		BU_ID
	PERSON_UID		PERSON_UID
	PRIV_FLG		PRIV_FLG

Figure 13. User Key Columns for the S_CONTACT_U1 User Key

If the columns of the user key are exposed in the business component and those columns are not foreign keys, 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. For example, in [Figure 14](#), the user key columns are exposed in the Integration Component Fields applet for the Contact integration component.

Business Components							
W	Name	Changed	Project	Cache Data	Class	Data Source	Dirty Reads
>	Contact	✓	Contact		CSSBCUser		✓

Fields						
Required	W	Name	Changed	Join	Column	PickList
>		Accomplishments		S_CONTACT_T	ACCOMPLISH	
		Account				
		Account Currency Code		Contact - S_ORG	BASE_CURCY_CI	
		Account Id		S_CONTACT	PR_DEPT_OU_ID	
		Account Integration Id		Contact - S_ORG	INTEGRATION_I	PickList Account

Figure 14. Integration Component Field List

About Integration Objects

Structure of Siebel Integration Objects

The Integration Object Builder wizard, for the preceding example, builds the integration component keys based on these table user keys. As illustrated in [Figure 15](#), the wizard defines one integration component key field 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		

Figure 15. Integration Component Keys for Each Table User Key Column

Each valid integration component key contains fields. For example, as shown in [Figure 16](#), for the Contact integration component, User Key 3 is made up of five fields: CSN, First Name, Last Name, Middle Name, and Personal Contact.

NOTE: You should 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

Figure 16. Contact Integration Component Key Fields

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 will help uniquely identify a given record. In some cases, you may 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

In the context of Siebel business objects, user keys are a group of fields whose values must uniquely identify only one Siebel business component record. Integration components within a corresponding integration object also contain user keys.

For many integrations, you want to know the status. For example, if you are sending an order request you want to know the ID of the Order created so that you can query on the order in the future. You can set the Status Object of EAI Siebel Adapter 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 Status Object. 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 or synchronize request, include a field named *Operation* in the Status Key.

Hierarchy Parent Key

The Hierarchy Parent Key is used for integration objects that have a homogeneous hierarchy. This key should only have 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 very first quote item inserted by EAI Siebel Adapter has the Parent Id set to blank, but for each child quote item, EAI Siebel Adapter checks the keys to figure out which fields are to be set. If 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).

Hierarchy Root Key

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 should have only one field, Root Id, which 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 need to set the property Hierarchy Parent Field.

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.

Integration Object Builder Overview

The Integration Object Builder 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, you should confirm that user key definitions are defined properly. You may need to enter keys and user properties manually or inactivate unused keys and fields in Siebel Tools. You should not expect to use the integration object without modification.

The following checklist gives the high-level procedure for creating an integration object.

Checklist

-
- ☐ Create integration objects using the EAI Siebel Wizard.
For details, see [“Creating Integration Objects Using the EAI Siebel Wizard” on page 48](#).
-

Checklist

-
- ☐ Fine-tune your integration object.
For details, see [“Siebel Integration Object Fine-Tuning”](#) on page 51.
 - ☐ Validate your integration object.
For details, see [“Integration Object Validation”](#) on page 51.
-

Creating Integration Objects Using the EAI Siebel Wizard

Siebel Tools provides a wizard to walk you through creating an integration object. You should use this wizard to create your integration object.

To create a new Siebel integration object

- 1** Start Siebel Tools.
- 2** Create a new project and lock it, or lock an existing project in which you want to create your integration object.
- 3** Choose File > New Object... to display the New Object Wizards dialog box.
- 4** Select the EAI tab and double-click the Integration Object icon.
- 5** In the Integration Object Builder wizard:
 - a** Select the project you locked in [Step 2](#).
 - b** Select the EAI Siebel Wizard.
- 6** Click Next and in the second page of the Integration Object Builder wizard:
 - a** Select the source object. This is the object model for the new Siebel integration object. Only business objects with Primary Business Components appear on this picklist.

- b** 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.

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

- 7** Deselect the components you would like the wizard to ignore. This means you will not be able to 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, the children below 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 top of the scrolling list. This action deselects the entire tree below Account.
 - b** Select the Contact component. When selecting a child component, its parent component is also selected, but none of the components below the child component are selected. You must individually select the ones you want.
- 8** 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.

- 9 Click Finish to complete the process of creating a new Siebel integration object.

NOTE: After creating integration objects in Siebel Tools, you must compile a new .srf file and copy the .srf file to the *SIEBSRVR_ROOT/OBJECTS* directory.

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: Once you create your integration object based on a Siebel business object, you should 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.

- 10 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 applet 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 EAI Siebel Adapter Query and QueryPage method from outputting these fields. For more details, see [“System Fields” on page 74](#) and [“Search Specification” on page 134](#).

NOTE: The XML Sequence property in this applet defines the order in which the XML tags appear when an integration object is created.

Siebel Integration Object Fine-Tuning

After you create your integration object you need to fine-tune and customize your integration object based on your business requirements. Following is a list of the most common practices in fine-tuning an integration object.

- Deactivate the fields that do not apply to your business requirements.
- If necessary, activate the fields that have been deactivated by the Siebel wizard. For details, see [Chapter 1, “About Integration Objects.”](#)
- Add the fields that have not been included by the Siebel wizard. For details on the implications of activating such fields, see [“Calculated Fields” on page 35](#) and [“Inner Joins” on page 35](#).
- Validate the user keys. For details, see [Chapter 1, “About Integration Objects.”](#)
- Update the user properties for your integration object to reflect your business requirements. For details, see [“Integration Object User Properties” on page 66](#).

Integration Object Validation

Once you have created your integration object and made the necessary modifications to meet your business requirements, you need to validate your integration object.

To validate your integration object

- 1** Open Siebel Tools.
- 2** Select your integration object.
- 3** Click Validate.

NOTE: Review the report and modify your integration object as needed. Integration objects you create in Siebel Tools must be compiled into the Siebel.srf file. Once you test the integration object, you must copy the compiled .srf to your `SIEBSRVR_ROOT\OBJECTS` directory.

Integration Objects Synchronization

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 applications

Synchronization Considerations

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

The following checklist gives the high-level steps for updating an integration object.

Checklist

- ☐ Run the Synchronization wizard.
For details, see [“Updating the Entire Integration Object” on page 57](#).
-

Checklist

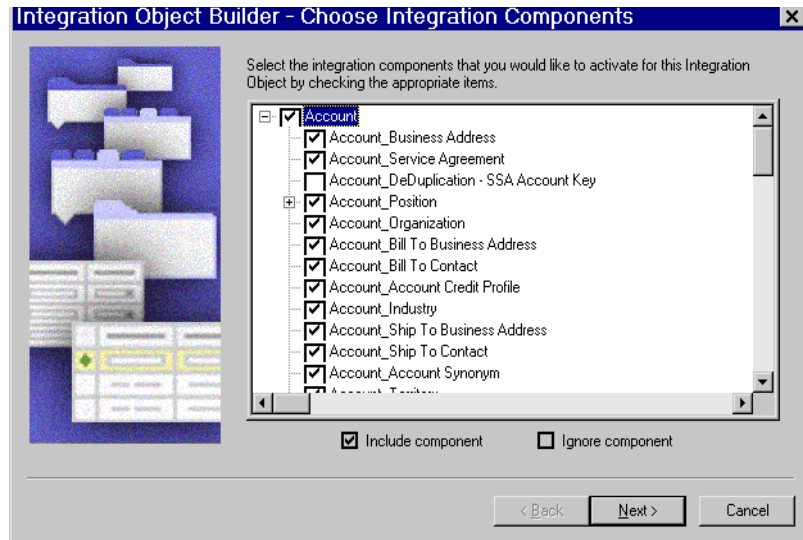
-
- ☐ Modify the newly updated integration object as needed, using the DIFF tool and the copy of the integration object as reference.
For details, see *Siebel Tools Reference*.
 - ☐ Run Validation.
For details, see [“Integration Object Validation” on page 51](#).
-

To update an integration object

- 1** Access the integration object you want to update in Siebel Tools.
- 2** Run the Synchronization wizard by double-clicking the Synchronization button.

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 modified the object.

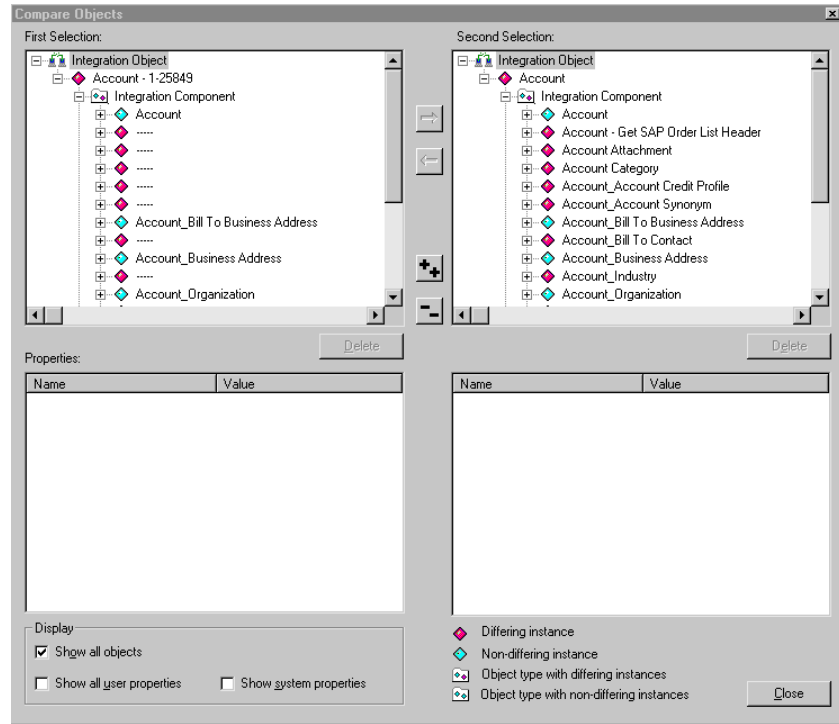
- a On the Integration Objects Builder, click on the plus sign to list all the related integration components, as shown in the following figure.



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

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

- c Click Finish to synchronize the Siebel integration object and the Siebel business object. The Compare Objects dialog box, shown below, 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. The Synchronization wizard selects or deselects components to make the Siebel integration object look like the representation 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 details, see [“Updating the Entire Integration Object” on page 57](#) and [“Deleting a Component from the Integration Object” on page 60](#).

- 3** 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.
- 4** Deactivate any new field that you do not need as a component of your updated integration object.
- 5** Right-click on your integration object, and select the Validate option to validate your integration object.

NOTE: If you need to synchronize any of the external integration objects, you should also follow this general procedure to perform a synchronization operation.

Synchronization Rules

During the synchronization process, the Synchronization 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. [Figure 17](#) helps you to visualize this example.

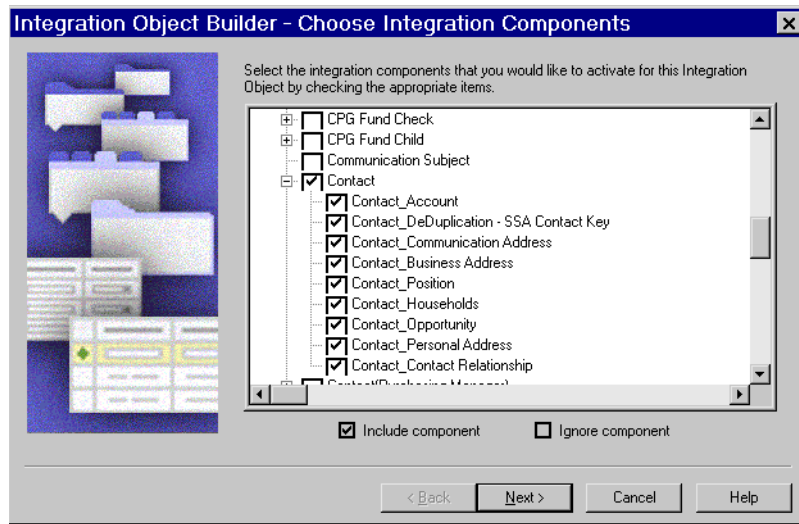


Figure 17. Example of Selected Integration Components

Since the Account component is the parent of Contact, it is also selected, even though you cannot see it in [Figure 17](#).

Updating the Entire Integration Object

After initiating the Synchronization wizard, if you check the boxes in the wizard, the wizard creates a new integration object in memory. If the underlying Siebel business object has been changed, then the new, in-memory integration object will be different from the integration object in the database. As a result, the wizard synchronizes the outdated integration object in the database with the new, in-memory integration object.

Figure 18 illustrates this concept.

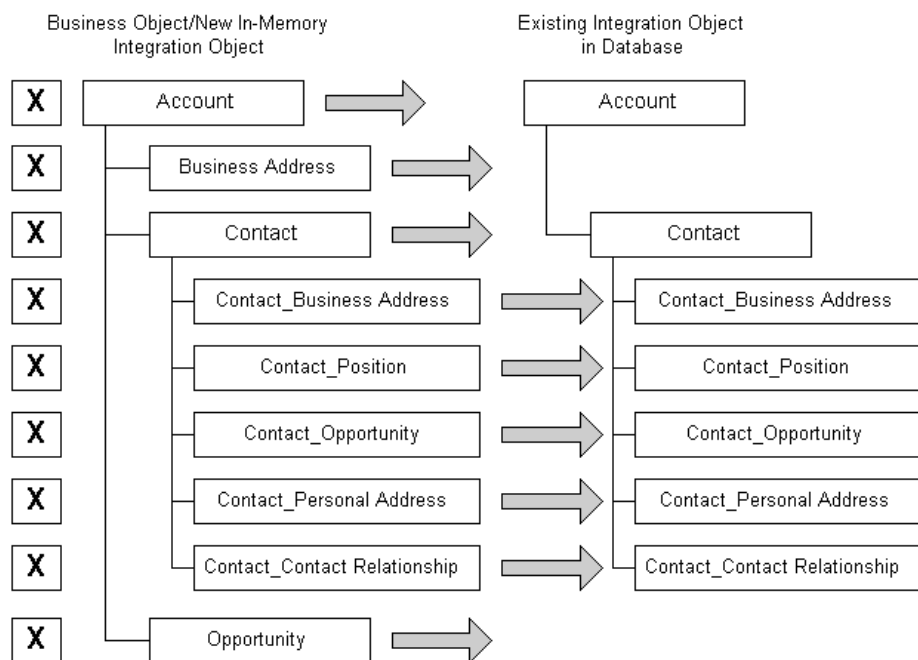


Figure 18. Synchronizing the Integration Object

Figure 19 shows how the resulting integration object is structured after the synchronization.

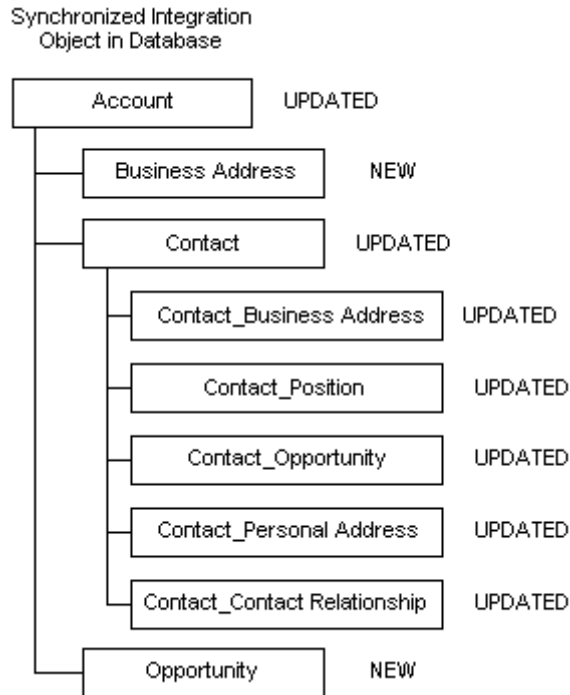


Figure 19. Completely Updated Integration Object

The integration object now contains two new components, *Business Address* and *Opportunity*. Other components have been 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, you specify to the wizard that it should 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 deselected component in the new, in-memory integration object.

Figure 20 illustrates this concept.

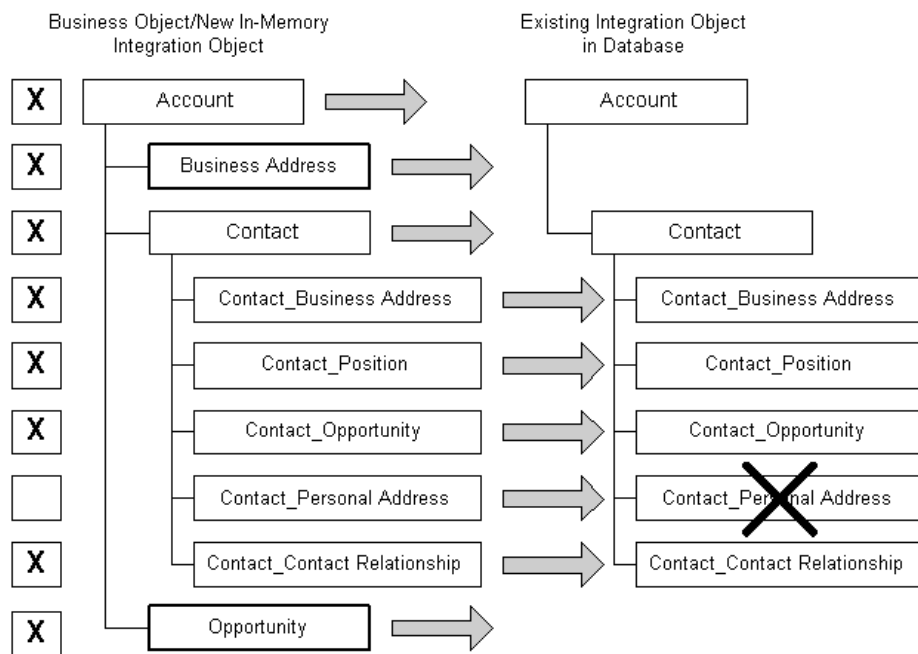


Figure 20. Deleting a Component from the Integration Object

Figure 21 shows the integration object after synchronization.

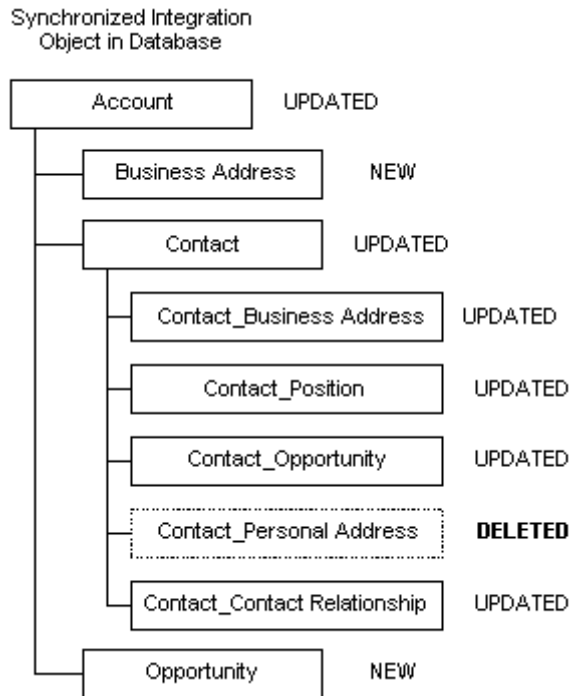


Figure 21. Synchronization Resulting in a Deleted Component

The component Contact_Personal Address has been deleted. When you use the updated integration object, you will not be able to pass data for that component between Siebel application and an external application.

This example is intended to show how you might cause unexpected results by deselecting components. However, if you do want to delete a particular component from the integration object, deleting a component from the integration object method accomplishes that goal.

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

The EAI Siebel Wizard

You can use the EAI Siebel Wizard to create integration objects that represent Siebel business objects. During the process of creating a new integration object, described in [“Integration Object Builder Overview” on page 47](#), 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. As shown in [Figure 22](#), the wizard returns a list of the available business objects from which you can choose one to base your integration object on.

The wizard also returns a list of the available components contained within the object you have chosen. When you select certain components in the wizard, you are activating those components in your integration object. Your integration object actually 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.

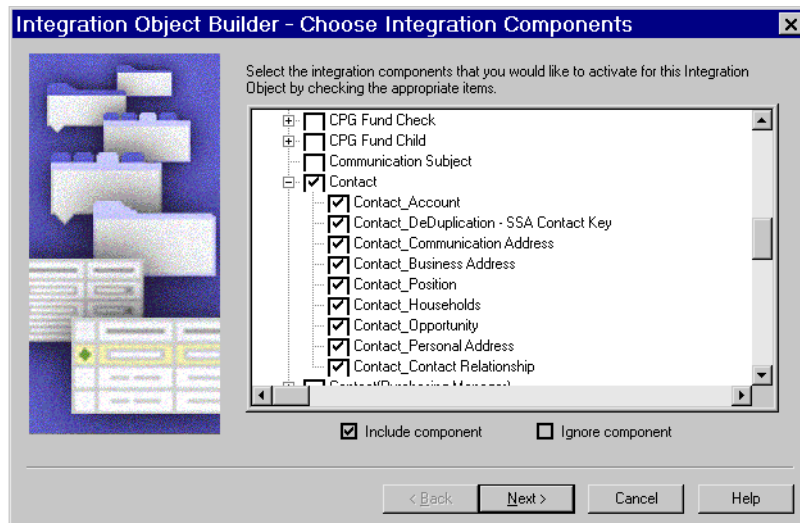


Figure 22. Activated Components in the Contact Integration Object

After the wizard creates your integration object, you can edit the object in Siebel Tools, as shown in [Figure 23](#). 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: You should always deactivate the fields rather than delete them, even though the net effect (as well as the DTD generated) will be the same. When you execute the synchronization task, using the Siebel eAI sync utility in Siebel Tools, inactivated fields remain inactive, while the deleted fields are created as active fields in the integration object.

Integration Components					
W	External Name Context	Name	Changed	Parent Integration Component	Extens
		Contact_Contact Relationship	✓	Contact	Cont
		Contact_DeDuplication - SSA Contact Key	✓	Contact	DeDu
		Contact_Households	✓	Contact	Hous
		Contact_Opportunity	✓	Contact	Opp
		Contact_Personal Address	✓	Contact	Persc

Integration Component Fields					
W	Name	Changed	Data Type	Length	Precision
		✓	DTYPE_TEXT	100	
		✓	DTYPE_TEXT	30	
		✓	DTYPE_TEXT	50	
		✓	DTYPE_TEXT	50	
		✓	DTYPE_TEXT	30	
		✓	DTYPE_TEXT	10	
		✓	DTYPE_TEXT	200	

Figure 23. Activated Components in the Contact Integration Object

Siebel Integration Objects Maintenance and Upgrade

Sometimes you may change the underlying business objects, which necessitates maintenance of the integration object. Synchronize the integration object by clicking the synchronize button.

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

- Name any user key that you add differently from the generated user keys. Using meaningful names helps with debugging.

- Inactivate user keys instead of deleting them.
- Inactivate fields instead of deleting them.

Permission Rules for Integration Components

Each business component, link, MVG, and integration object user property has settings such as No Update, No Delete, and No Insert. These settings indicate the type of operations that cannot be performed on that object. In order for EAI Siebel Adapter to successfully perform an operation, that operation needs to be allowed at all levels. If the operation is allowed at every level but the field level, a warning message is logged in the log file and processing continues. Otherwise, an error message is returned and the transaction is rolled back.

Table 5 illustrates which permissions influence which operation type on an integration component.

Table 5. Permission Rules for an Integration Component

Permission Layer	Checked by...	Integration Component Type		
		Standard	MVG	Association
Integration Object Component	EAI Siebel Adapter	✓	✓	✓
Integration Component		✓	✓	✓
Integration Field (Update Only)		✓	✓	✓
Link	Object Manager	✓	✓	✓
Multi-Value Link (MVL)			✓	
Business Component (Overridden by AdminMode)		✓	✓	✓
Business Component Field		✓	✓	✓

NOTE: The transaction is rolled back if any of the permissions (excluding field-level permissions) are denied.

EAI Siebel Adapter Access Control

You can use the following mechanisms to control EAI Siebel Adapter access 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, 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 EAI Siebel Adapter can use. This allows you to minimize the number of integration objects your users need to expose outside of Siebel applications through HTTP inbound or MQSeries Receiver server components. If this user property is not specified, EAI Siebel Adapter uses any integration objects defined in the current Siebel Repository.
- **ViewMode.** You can specify the visibility mode of business components that 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 details on ViewMode, see *Siebel Tools Online Help*.

Integration Object User Properties

You can define user properties for your integration objects. These user properties help determine special processing and behavioral requirements of integration objects for a specific eAI adapter.

The Level column shown in [Table 6](#) can take on the following values:

- O, for Object Level
- C, for Component Level

- F, for Field Level

Table 6. Integration Objects User Properties

User Property	Allowable Values	Level	Description
Association	Y,N	C	Default is N. If set to Y, it labels the integration component as having a many-to-many relationship with its parent integration component configured by a Link with an intersection table. Not applicable to root integration component.
MVG	Y,N	C	Default is N. If set to Y, it labels the integration component as having a many-to-one relationship with its parent integration component configured by a Multi Value Link defined over a Link without an intersection table. Not applicable to root integration component.
Picklist	Y,N	F	Default is N. If set to Y, the field is based on a picklist and the EAI Siebel Adapter validates the field value using an associated picklist, bounded or non-bounded. If the picklist is non-bounded and the value does not match, then the EAI Siebel Adapter logs a warning but the value is set accordingly. If this property is set to N, or is not defined, then the EAI Siebel Adapter leaves the validation to the Object Manager, aborts the processing if the validation fails for the bounded picklist, and logs a warning. See “Performance Considerations” on page 73 .
PicklistUserKeys		F	The value of this 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). For details, see “Picklists” on page 33 .
Ignore Bounded Picklist	Y,N	O,C, F	Default is N. If this property is set to N and the Picklist is set to Y, and the value provided does not match any of the values in the picklist, then the EAI Siebel Adapter stops processing, writes an error to the log file, and rolls back the transaction. If this property is set to Y and the value provided does not match any of the values in the picklist, then the EAI Siebel Adapter reports a warning in the log file and sets the field to Null.

Table 6. Integration Objects User Properties

User Property	Allowable Values	Level	Description
MVGAssociation	Y,N	C	Default is N. If set to Y, it means the integration component has a many-to-many relationship with its parent integration component configured by a Multi-Value Link defined over a Link with an intersection table. Not applicable to the root integration component.
MVGFieldName	Any valid field name in the business component	F	If the component that owns this integration field is labeled MVG, this user property gives the name of the business component field as the value known by the MVG component. In this case, the External Name property of the integration component field references the field name in the parent business component.
AssocFieldName	Any valid field name in the business component	F	If the component that owns this integration field is labeled MVGAssociation, this user property gives the name of the business component field as the value known by the Association MVG component. In this case, the External Name property of integration component field references the field name in the parent business component.
NoInsert	Y,N	C	Default is N. If this property is set to Y at the component level, the EAI Siebel Adapter is prevented from inserting a new record into the component. The EAI Siebel Adapter aborts the processing of the EAI Message and returns an error message. This allows you to limit the functionality of an integration object to a subset of what the underlying business object allows. This user property cannot override limitations imposed by the Business Object, Business Component, Link, and Multi-Value-Link.
NoDelete	Y,N	C	Default is N. If this property is set to Y at the component level, no records in that component can be deleted. The EAI Siebel Adapter aborts the processing of the message and returns an error message. This allows you to limit the functionality of the integration object to a subset of what the underlying business object allows. This user property cannot override limitations imposed by the business object, business component, Link, and Multi-Value-Link.

Table 6. Integration Objects User Properties

User Property	Allowable Values	Level	Description
NoUpdate	Y,N	C,F	Default is N. If this property is set to Y at the Component level, no fields in that component can be updated. If an existing record needs to be updated, EAI Siebel Adapter aborts the processing of the EAI Message and returns an error message. If this property is set to Y on the field level, then EAI Siebel Adapter only logs a warning message and skips updating the field, and continues processing. This allows you to limit the functionality of the integration object to a subset of what the underlying business object allows. This user property cannot override limitations imposed by the business object, business component, Link, Multi-Value-Link, or Field.
FieldDependencyXXX	Any active integration component field name within the same integration component	F	Defines a dependency between the integration field that has this user property and the integration field specified by the value of the user property. Multiple dependencies are specified by separate user property entries. The field specified in the value must be from the same component as the field that has the user property. Dependencies constrain the order of field processing within an integration component. If field A depends on field B, then field B is processed before field A. Also see Chapter 1, “About Integration Objects.”
AdminMode	Y,N	C,O	Default is N. Sets AdminMode on the business component. Some business components, such as Internal Product, allow only administrators to make modifications. You may allow modification of such components during integration, by setting the AdminMode property to Y on the integration component or the integration object level. The setting at the integration component level overrides the setting at the business object level. This property can not be used in MVGs. For details, see <i>Siebel Tools Online Help</i>

Table 6. Integration Objects User Properties

User Property	Allowable Values	Level	Description
ViewMode	Manager, Sales Rep, Personal, Catalog, Group, Organization, All	O	Default is All. Specifies the visibility mode of the business component that EAI Siebel Adapter uses. The allowable values are based on REPOSITORY_BC_VIEWMODE_TYPE LOV.
AllLangIndependent Vals	Y,N	O	Default is N. If set to Y, this user property forces the EAI Siebel Adapter to use language-independent values for LOV-based integration component fields. This is useful when there is a requirement to support integration between systems that use multiple languages. If set to N, all LOV-based fields use language-dependent values. If this user property is not defined for the integration object, multilingual LOV-based fields (MLOV) use language independent values, while single-language LOV fields use language dependent values.

Example of an Integration Object With M:M Relationship

Following is an example of how to create an integration object with two components that have a many-to-many (M:M) relationship. For illustration purposes, we are using an integration object that uses Contact business object with Contact and Opportunity business components.

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

- 1 Start Siebel Tools.
- 2 Create a new project and lock it, or lock an existing project in which you want to create your integration object.
- 3 Choose File > New Object... to display the New Object Wizards dialog box.
- 4 Select the EAI tab and double-click the Integration Object icon.
- 5 In the Integration Object Builder wizard:
 - a Select the project you locked in [Step 2](#).

- b** Select the EAI Siebel Wizard.
- 6** 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 and click Next—for example, Sample Contact M:M.
- 7** From the list of components, select Contact and Opportunity. There is also a component named Contact_Opportunity in the list. This component is an MVGAssociation component, and should be picked only if you need this integration object to set the primary opportunity for contact. For details on MVG, see [“MVGs in EAI Siebel Adapter” on page 131](#).
- 8** Deactivate 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 we need for Contact.)
- 9** Deactivate 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 we need for Opportunity.)
- 10** Compile a new .srf file and copy the .srf file to the *SIEBSRVR_ROOT/OBJECTS* directory.

To test the newly created integration object

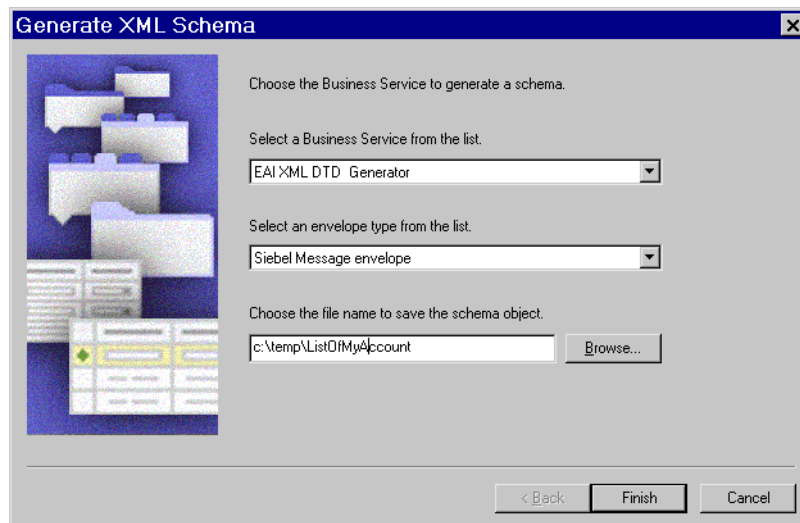
- 1** Start Siebel client connected to Sample database.
- 2** Copy and modify the Import Account (File) and the Export Account (File) sample workflow processes to work with the Contact business object, instead of the Account business object.
- 3** Modify the Export Account (File) workflow process to invoke the EAI Siebel Adapter against the Sample Contact M:M integration object that you created in [“To create an integration object with a many-to-many business component” on page 70](#).
- 4** Run the workflow processes using the Workflow Process Simulator.

Generating Schemas

At certain points in your integration project, you may want to generate schemas from an integration object. If you export Siebel integration objects as XML to other applications, you may need 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, click on an integration object to make it the active object.
- 2 Click Generate Schema to access the Generate XML Schema wizard shown in the following figure.,



- 3 Choose the EAI XML DTD Generator business service.
- 4 Choose an envelope type to use in generated DTD.
- 5 Choose a location where you want to save the resulting DTD file and click Finish. The wizard generates a DTD of the integration object you selected. Use this DTD to help you map external data directly to the integration object. The DTD serves as the definition for the XML elements you can create using an external application or XML editing tool.

Performance Considerations

To optimize your integration object performance, you may 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 EAI Siebel Adapter operations. You should inactivate unnecessary fields and components in your integration objects.

Force-Active Fields

You should reexamine any fields in the underlying business component that are force-active. Such fields are processed during 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 applications have two classes of picklists, static picklists based on list 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 may be unnecessary and can be turned off by setting the PICKLIST property to N. Even for bounded static picklists, validation in the adapter can be turned off 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 will 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.

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

Business Component Restrictions

The business components underlying the Integration Components may have certain restrictions. For example, Internal Product can only be modified by an administrator. 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 Query and QueryPage method the EAI Siebel Adapter will not output 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.

NOTE: If you want to include System fields in the exported message, change the Integration Component field type to Data. System fields are read only. If you attempt to send in a message with the value set for a System field, the setting will be ignored and a warning message will be logged.

Best Practices

- Familiarize yourself with the business logic in the business components. Integration designers should 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. See [“Performance Considerations” on page 73](#).
- Design with maintenance in mind. See [“Siebel Integration Objects Maintenance and Upgrade” on page 64](#).

- 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 three different situations in which you might encounter such conflicts and a possible solution for each case.

Scenario 1. Your integration requires explicitly setting a primary child, but the business component configuration does not allow that because the related MVLink has Auto Primary property set to Default.

Solution. Change the Auto Primary property from Default to Selected. This enables EAI Siebel Adapter to change the Auto Primary property according to the input request, while making sure that there is always a primary child selected.

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.

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 3. 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. Since 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 need to 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 the same business component should be used by both links.

NOTE: Customized configurations are not automatically upgraded during the Siebel Repository upgrade, so this option should be used as a last resort.

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.

Overview of 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.

These services can then be accessed by Siebel VB or Siebel eScript code that you write and call from workflow processes. For the purposes of your integration projects using Siebel eAI, you can use Siebel eScript to write your scripts to use the DTE scripts.

Creating Business Services

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

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

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 repository (*.srf), so you have to compile the repository before testing them. Once your test is completed, you need to compile and disseminate the .srf to your clients. The business services stored in the repository will 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 CSSEAITEScriptService class. For details, see [“Creating Business Services in Siebel Tools” on page 82](#).
- **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 Database, so they can be tested right away. The run-time business services have to be manually moved over after an upgrade process. For details, see [“Creating a Business Service in the Siebel Client” on page 87](#).

NOTE: To use the DTE scripts, you need to write your business service in 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 may act upon. A business service is much like an object in an object-oriented programming language.

A 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 details on business service methods and method arguments, see *Siebel Tools Online Help*.

About 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. [Figure 24](#) illustrates the concept of a property set.

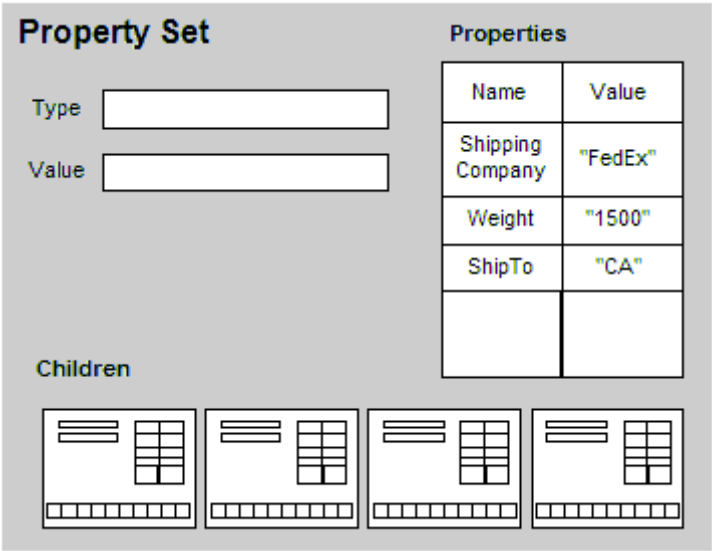


Figure 24. Property Set Structure

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 brackets. You can also define a Display Name for the Value argument in the Business Service Simulator. This Display Name appears in the Workflow 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. The properties can be used 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. The array can be used to represent instances of integration objects; for example, a result set may contain an Account with some set of contact records from the database. Each contact record is represented as a child property set.

NOTE: For details on property sets and their methods, see *Siebel Tools Online Help*.

Creating Business Services in Siebel Tools

The following sections explain how to create business services and business service scripts in Siebel Tools.

Checklist

- ☐ Define the Business Service
For details, see [“To define a business service in Siebel Tools” on page 83.](#)
 - ☐ Define the Business Service Methods
For details, see [“To define a business service method” on page 84.](#)
 - ☐ Define the Business Service Methods Arguments
For details, see [“To define the business service method arguments” on page 84.](#)
 - ☐ Define Business Service Scripts
For details, see [“To define and write the business service script” on page 85.](#)
 - ☐ Define Business Service Subsystem
For details, see [“To specify a business service subsystem” on page 86.](#)
 - ☐ Define Business Service User Properties
For details, see [“To define business service user properties” on page 87.](#)
-

NOTE: Business services you create in Siebel Tools must be compiled into the Siebel .srf file. If you intend to run the business services on your Siebel Server, then copy the compiled .srf file to your *SIEBSRVR_ROOT\Object\lang* directory.

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 Script Editor.

To define a business service in Siebel Tools

- 1** Start Siebel Tools.
- 2** Select and lock the project you want to associate your business service with.

NOTE: Each business service must belong to a project and the project must be locked. For details, see *Siebel Tools Reference*.

- 3** Select the EAI Business Services object in the Tools Object Explorer.
The list of predefined business services appears in the right panel.
- 4** Choose Edit New Record to create a new business service.
- 5** Type a name for your business service in the Name field.
- 6** Type the name of the project you locked in [Step 2](#), in the Project field.
- 7** Choose the appropriate class for your business service, from the Class picklist.
 - Data transformation business services should use the CSSEAITEScriptService class.
 - Other business services will typically use the CSSService class.
- 8** 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 details on business service methods, see *Siebel Tools Online Help*.

To define a business service method

- 1 With your business service selected, double-click the Business Services Methods folder in the Siebel Tools Object Explorer.

The Business Services Methods list appears below the list of business services. If you have already defined methods for the selected business service, the method names appear in the Business Services Methods list.

- 2 Choose Edit > New Record to create a new method.
- 3 Type the name of the method in the Name field.

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 the business service method arguments

- 1 With your business service selected, double-click the Business Service Method Arg folder, in the Tools Object Explorer, to display the Business Service Method Args list.
- 2 Choose Edit > New Record to create a blank method argument record.
- 3 Type the name of the argument in the Name field.

NOTE: If you plan to use this business service in a Siebel Client, you need to 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.

Defining and 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 define and write the business service script

- 1 Start Siebel Tools.
- 2 Select the business service for which you want to write a script.
- 3 Right-click to display a pop-up menu.
- 4 Choose Edit Server Scripts.
- 5 Select either eScript or Visual Basic for your scripting language.

Service-PreInvokedMethod is selected as the service.

NOTE: To write any Siebel VB script in the Business Services, your deployment platform must support Siebel VB.

6 Type your script into the Script Editor.

NOTE: You need to write your business service in eScript if you want to use the DTE scripts. For details on scripting, see *Siebel Tools Online Help*.

Specifying Business Service Subsystems

You can optionally specify a business service subsystem. A business service subsystem is a server component that encapsulates a large amount of functionality and that is already included in the Siebel repository. Business service subsystems define particular events upon which the subsystem will be called. The subsystems can also trigger other events, depending on how they are defined. Examples of business service subsystems are presented in [Table 7](#).

Table 7. Business Service Subsystems

Subsystem	Description
EAISubsys	Defines events for a variety of eAI operations, including the initiation of eAI wizards, calls to eAI adapters, and calls to eAI validation routines.
SAPSubsys	Defines a variety of parameters to help determine the type of SAP object being integrated, the transport mechanism, user name and password combinations, and SAP program ID.
Workflow	Defines both events and parameters to signal and determine behaviors based on the initiation of workflow processes, search specifications, and Row Id.
XMLCnv	Defines events regarding debugging information and responses from the XML parser.

To specify a business service subsystem

- 1 With your business service selected, double-click the Business Service Subsystem folder in the Tools Object Explorer to display a list of subsystems.
- 2 Choose Edit > New Record to create a blank business service subsystem record.
- 3 Choose an existing business service subsystem name from the Subsystem picklist.

Defining Business Service User Properties

User properties, also known as User Props, are optional variables that you can use to define default values for your business services. When a script or control invokes 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, double-click the Business Service User Prop folder in the Tools Object Explorer to display the list of Business Service User Props.
- 2 Choose Edit > New Record to create a blank user property record.
- 3 Type the name of the user property in the Name field.
- 4 Type a value in the Value field.

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

Creating a Business Service in the Siebel Client

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

To define a business service in the Siebel Client

- 1 From the application-level menu, choose View > Site Map > Business Service Administration > Business Service Methods.

- 2 Click New to create a new record in the Business Service list 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 be optional.

Inactive. Check if you do not want to use the argument.

- 5 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, your deployment platform must support Siebel VB.

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

Business Service Export and Import

Business services can be exported into an XML file by clicking the Export Service button in the Business Service list applet. This writes the definition of the business service including every method, method argument, and script into the XML file.

You can also import a business service from an external XML file by clicking the Import Service button in the Business Service list applet.

Testing Your Business Service

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

To run the Business Service Simulator

- 1 From the application-level menu, choose View > Site Map > Business Service Administration > Business Service Simulator.

NOTE: The contents of the Simulator screen 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 Service Methods 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, 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, 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: Once the Output arguments are created, you can click Move To Input to test the outputs as inputs to another method.

Accessing a Business Service Using Siebel eScript or Siebel VB

In addition to accessing a business service through a workflow process, 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 produce a property set as an output. The output property set is used by EAI Siebel Adapter 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  
msgbox Cstr(XMLOutputs.GetChildCount)
```

Business Scenario

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 may 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 may 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

An example of the code you need to write to create the property set may look something like this:

```
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);
    }
}
```

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.

Web Services Overview

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

Web Services can be implemented in Siebel eBusiness applications as business services or workflow processes. The Siebel Web Services Framework can consume a WSDL document and create a proxy business service through the WSDL Import Wizard provided 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

The following are the Web Services standards supported by Siebel application:

- Web Services Description Language (WSDL) 1.1. For details, see <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>.
- Simple Object Access Protocol (SOAP) 1.1. For details, see <http://www.w3.org/TR/2000/NOTE-SOAP-20000508>.
- Hypertext Transfer Protocol -- HTTP/1.0. For details, see <http://www.w3.org/Protocols/rfc1945/rfc1945>.
- Extensible Markup Language (XML) 1.0. For details, see <http://www.w3.org/TR/1998/REC-xml-19980210>
- XML Schema. For details, see <http://www.w3.org/TR/2001/REC-xmlschema-1-20010502>, and <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502>

NOTE: For more details on supported elements and attributes, see *XML Reference: Siebel eBusiness Application Integration Volume V*.

Web Services Inbound Dispatcher Defined

The Web Service Inbound Dispatcher is a business service that is called by an inbound transport server component (or an outbound Web Service dispatcher) and analyzes input XML, converts XML data to business service method arguments, and calls the appropriate method for the appropriate service. After the called method has finished, the Web Service Inbound Dispatcher converts the output arguments to XML data, or creates a SOAP fault block (if there is an exception) and then returns the XML embedded in the SOAP envelope.

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 are examples of input arguments.

Name: siebel_transport_header:UserDefinedHeader

Value: myData

Name: siebel_transport_header:Authorization

Value: 0135DFDJKLJ

Inbound Web Services

The Inbound Web Service allows an external system to call a Siebel published Web Service. You can publish a business service or a business process as a Web Service and generate a Web Service Definition Language (WSDL) file that an external system can import. The Inbound Web Services can only be published from Siebel C using SOAP-RPC binding.

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.

To create a new Inbound Web Service record

- 1** From the application-level menu, choose View > Site Map > Web Services Administration > Inbound Web Services view.
- 2** In the Inbound Web Services list applet, create a new Inbound Web Services record.
 - a** Enter the namespace for your organization's Web Services in the Namespace column.

- b** Enter the name of the inbound Web Service in the Name column.
- c** Select Active or Inactive in the Status field.

NOTE: If the Web Service is inactive, then the external applications cannot invoke the Web Service. If the status is changed, the server component running the inbound transport, such as HTTP, requires a restart for the change to take effect.

- 3** Enter a description of the Web Service in the Comment column.
- 4** Create a new inbound service port record in the Service Ports list applet.
 - a** Enter the name of the port in the Name column.
 - b** Pick the type of object published.

If the required type is not available, add a new type following [Step c on page 98](#) through [Step f on page 98](#); otherwise, move to [Step g on page 98](#).
 - c** Click New and select the implementation type (Business Service or Workflow).
 - 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 Pick Applet to complete the process of adding a new Type.
 - g** Select the protocol or transport to publish the Web Service on.
 - h** Enter the URL or queue to publish the Web Service on.

NOTE: When publishing over EAI MQSeries or EAI MSMQ, you cannot generate WSDL files.

- ☐ The format to publish over EAI MQSeries or EAI MSMQ Server transports is:

`mq://send receive service point name@policy name`

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

- The URL format to publish over HTTP is:

```
http://webserver/eai_lang/  
start.swe?SWEExtSource=WebService&SWEExtCmd=Execute&UserNa  
me=username&Password=password
```

Where:

lang is the default language of Object Manager to handle the request.

webserver is the machine name of the Siebel Web Server.

username is the Siebel user to execute the request.

password is the password of the Siebel user.

NOTE: The Siebel application supports only one type of binding, SOAP_RPC, for each Inbound Web Service.

- 5 Enter a description of the Port in the Comment column.
- 6 In the Operations list applet, create a new operation record for the new service port you created in [Step 4 on page 98](#) and want to publish.

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 Business Service Method column to complete the process.

NOTE: The Business Service Method column defaults to RunProcess if you have chosen Workflow Process in [Step 4 on page 98](#) as the Type for your Service Port.

Generating a WSDL File

Once 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 Choose the inbound Web Services you want to publish and click GenerateWSDL.

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 VisualStudio.Net, use the wsdl.exe utility—for example, `wsdl.exe /l:CS mywsdlfile.wsdl`.
- In Apache's AXIS, use the wsdl2java utility—for example, `java org.apache.axis.wsdl.WSDL2Java mywsdlfile.wsdl`.
- In IBM's WSADIE, add the WSDL file to the Services perspective and run the Create Service Proxy wizard.

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

Outbound Web Services

An outbound Web Service definition acts as a proxy to a Web Service published by an external application. The outbound Web Service can be based on one of the following:

- External Web Service definition (WSDL) file
- Outbound Application Service Interface (ASI)

Outbound Web Services Based on an External WSDL File

The following procedure describes how to use the WSDL Import Wizard to read an external WSDL document.

To read an external WSDL document

- 1** Start Siebel Tools.
- 2** Create a new project and lock the project, or lock an existing project in which you want to create your integration object.
- 3** Choose File > New Object... to display the New Object Wizards.
- 4** Select the EAI tab, select the Web Service icon, and click OK.

The WSDL Import Wizard appears.
- 5** Select the Project where you want the objects to be held after they are created from the WSDL document.
- 6** Specify the WSDL document that contains the Web Service or Web Services definition that you want to import.
- 7** Specify the file where you want to store the run-time data extracted from the WSDL document.
- 8** Specify the log file where you want errors, warnings, and other information related to the import process to be logged.
- 9** Click Next to view and verify a summary of your import information.
- 10** Click Finish to complete the process of importing the business service into the Siebel repository.

This procedure generates three objects in Siebel repository.

- An outbound proxy business service of CSSWSOutboundDisptacher class.

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.

- One or more integration objects representing input and output parameters of the service methods.

- An XML document containing the run-time parameters that should be imported into the Siebel client. For details, see [“To import run-time data about external Web Service” on page 102](#).

Outbound Web Services Administration

The WSDL Import Wizard exports the data to a file that you must import to the run-time database (the Web Services address) using the Outbound Web Services screen.

To import run-time data about external Web Service

- 1 Restart the Siebel Server (or Mobile Web Client) with a recompiled version of the .srf file that includes the new objects created by the Web Services Import Wizard.

NOTE: You do not need to update your .srf file at design time. However, the service definition must exist in the .srf file during run time.

- 2 From the application-level menu, choose View > Site Map > Web Services Administration > Outbound Web Services view.
- 3 In the Outbound Web Services list applet, click Import to bring up the EAI Web Service Import dialog box.
- 4 Specify the export file created by the Web Services Import Wizard.
- 5 Click Import to import the Web Service definition into the database.

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, 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 From the application-level menu, choose View > Site Map > Web Services Administration > 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.
- 3** Enter a description of the Web Service in the Comment column.

NOTE: When importing an external Web Service, you do not need to specify the proxy business service, integration objects, or the run-time parameters.

- 4** 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 type of proxy for the Port Type column.
 - c** Select a transport name for the protocol or queuing system for the Transport.
 - d** Enter the address appropriate for the transport chosen.
 - ☐ For the Local Workflow or the Local Business Service transports, enter the name of a Business Process or Business Service that should be called.
 - ☐ For the Local Web Service transport, enter the name of the inbound port.
 - ☐ For the HTTP Transport, enter an HTTP address of the Web Service to be called—for example, `http://mycompany.com/webserivice/orderservice`.
 - ☐ For the EAI MQSeries AMI or EAI MSMQ Server transports, enter one of the following:

`mq://send receive service point name@policy name`

`msmq://queue name@queue machine name`
- 5** Select whether the port uses SOAP document, SOAP RPC, or property set Binding.

NOTE: Property Set Binding should be used 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 Local Workflow or Local Business Service transport to avoid overhead of processing XML.

- 6** Enter a description of the Port in the Comment column.
- 7** In the Operations Bindings applet, create a new Operations record.
 - a** Enter the name of the Web Service in the Name column.
 - b** Enter the name of the Binding Property in the Binding Property column; for example, SOAPAction.
 - c** Enter the value of the Binding Property in the Binding Value column; for example, CreateOrder.
- 8** Generate the WSDL file. For details, see [“To generate a WSDL file” on page 100](#).

Once you have created your outbound Web Service, you need to 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 need to take to accomplish this task.

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

- 1** Open Siebel Tools.
- 2** Select the outbound Web Service proxy business service you want to use to call your outbound Web Service.
- 3** 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 an Outbound Web Service

The property set that is used as an input argument to the outbound Web Service should have the same name as the input argument's name of the 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. You need to change the output from your business services in Siebel Tools, as well as the name of the property set child that contains integration object instance.
- Change the property set name from SiebelMessage to the actual outbound Web Service argument name by using an eScript service before calling the outbound Web Service.

XML Schema Support for <xsd:any> Tag

In the current framework, WSDL Import Wizard makes use of 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 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 an attribute called *namespace*. If the value for that attribute is known, then one or more integration components or even an integration object can be created. If not known, then the business service method argument for that particular <wsdl:part> tag will be changed to data type Hierarchy, consequently losing any type information.

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

The <xsd:anyAttribute> tag has an attribute called *namespace*. If the namespace value is known (the conditions for being known were noted in this section), 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 an attribute called *namespace*. If the value for that attribute is known, then one or more integration components or even an integration object can be created. If not, an error will be returned to the user saying that the integration object cannot be created for a weakly typed schema.

Being known refers to this situation for 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 an attribute called *namespace*. If the namespace value is known (the condition for being known was noted in this section), 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 invoke an external Web Service from a Siebel application or how to invoke a Siebel Web Service from an external application.

Invoking an External Web Service Using Workflow or Scripting

As illustrated on [Figure 25 on page 108](#), the following steps are executed to invoke an external Web Service.

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

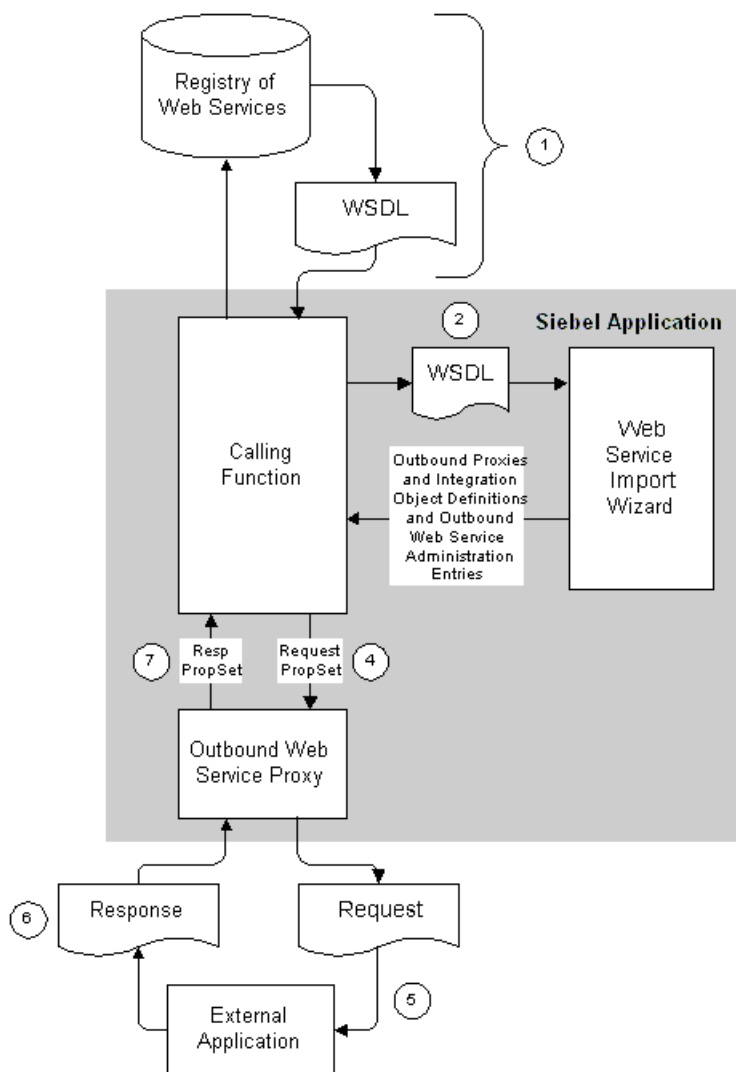


Figure 25. Invoking an External Web Service

The following example shows how to invoke Web Services using 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 [Figure 26 on page 113](#), 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 Siebel Inbound Web Services screen. 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 in order to create a proxy that can be used to call the Siebel Web Service from [Step 1](#).
- 3** The external application sends SOAP request into 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 gets returned from business service or business process to the Web Service Inbound Dispatcher.
- 6** Response is converted to a SOAP message and sent back to the calling external application.

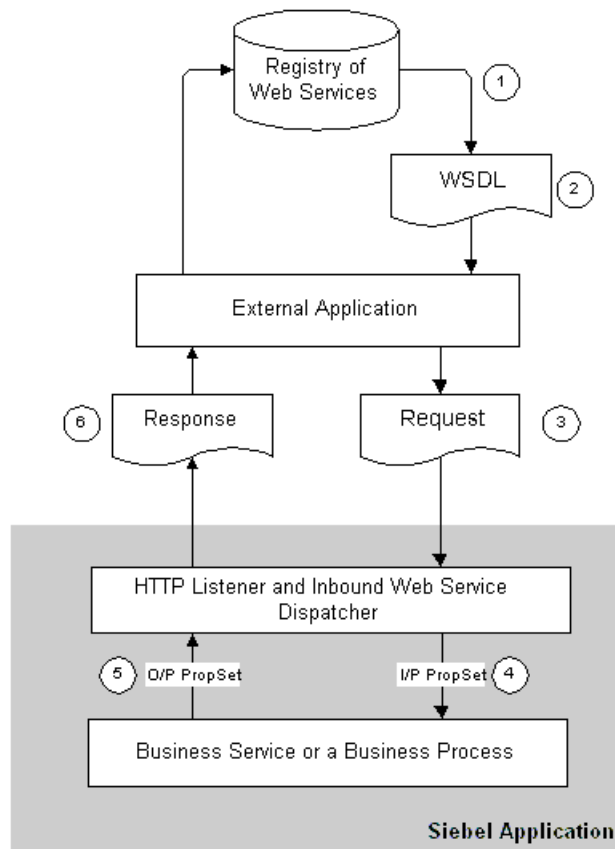


Figure 26. Invoking a Siebel Web Service

The following is an example of invoking 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.
// Since 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, I chose to run the wsdl.exe
// utility 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 additional checking 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;
}
}
}
```

Troubleshooting Tips

You can enable Web Services Tracing on the Server to write all inbound and outbound SOAP documents to a log file.

To enable Web Services Tracing

- 1** From the application-level menu, choose View > Site Map > Server Administration.
- 2** Go to the Servers view.
- 3** Select the Server Event Configuration tab.
- 4** Set the Log Level parameter to 4 for the following Event Types:
 - Web Service Inbound Argument Tracing
 - Web Service Outbound Argument Tracing
 - Web Service Inbound
 - Web Service Outbound
- 5** Navigate to the Components view.
- 6** Select the EAI Object Manager component, and select the Component Parameters tab.
- 7** Set the Enable Business Service Argument Tracing parameter to True.

- 8 Restart or reconfigure the server component. For details, see *Siebel Server Administration Guide*.

Integration Components Cardinality

The cardinality of the root integration component used by inbound Web Services has to be set to *Zero or More*. Cardinality of other integration components is not restricted.

The reason for the constraint on root component cardinality is that Siebel Web Services infrastructure generally returns multiple instances of root integration component for any given request. Thus, having cardinality set to anything other than *Zero or More* would prevent external clients to correctly interoperate with Siebel Web Services.

NOTE: When modifying run-time parameters, the server component needs to be restarted. For details, see *Siebel Server Administration Guide*.

This chapter describes the functionality of the EAI Siebel Adapter and the different methods and arguments you can use with the EAI Siebel Adapter to manipulate the data in the Siebel Database.

EAI Siebel Adapter Overview

The 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.
- Write an integration object whose data originates externally into a Siebel business object.
- Update multiple corresponding top-level parent business component records with data from one XML file—for examples, see [“XML Examples” on page 129](#).

NOTE: EAI Message is considered to be one transaction. The transaction is committed when there is no error. If there is an error, the transaction is aborted and rolled back.

The EAI Siebel Adapter business service is implemented by the class *CSSEAISiebelAdapter* that inherits from the *CSSService* class.

EAI Siebel Adapter Methods

The EAI Siebel Adapter uses *DoInvokeMethod* in order to provide an interface that performs the following methods:

- Query

- QueryPage
- Synchronize
- Upsert
- Insert
- Update
- Delete
- Execute

The implementation of *DoInvokeMethod* creates *CSSEAIMessageIn* and *CSSEAIMessageOut* objects by parsing the input property sets and invokes *Execute*, which does the right thing depending on the method. If an output is generated, it is stored into the *CSSEAIMessageOut* object.

```
class CSSEAISiebelAdapter : public CSSService
{
public:
    BOOLCanInvokeMethod(LPCSTR methodName);
    ErrCodeDoInvokeMethod(LPCSTR methodName,
                          const CSSPropertySetEx& inArgs,
                          CSSPropertySetEx& outArgs);
protected:
    ErrCodeExecute(CSSEAIMessageIn* pObjInst,
                  CSSEAIMessageOut*& pOutObjInst);
};
```

EAI Siebel Adapter 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 in [Table 8](#).

Table 8. EAI Siebel Adapter Method Arguments

Argument	Query	QueryPage	Sync	Upsert	Update	Insert	Delete	Execute
IntObjectName	-	-	-	-	-	'	“	Input
NumOutputObjects	Output	Output	Output	Output	Output	Output	Output	Output
OutputIntObjectName	Input	Input	-	-	-	-	-	Input
PrimaryRowId	Input	-	Output	Output	Output	Output	Input	Input/ Output
QueryByUserKey	Input	-	-	-	-	-	-	Input
DeleteByUserKey	-	-	-	-	-	-	Input	Input
ErrorOnNonExistingDelete	-	-	-	-	-	-	Input	Input
SiebelMessage	Input/ Output	Input/ Output	Input/ Output	Input/ Output	Input/ Output	Input/ Output	Input/ Output	Input/ Output
SearchSpec	Input	Input	-	-	-	-	Input	Input
StatusObject	-	-	Input	Input	Input	Input	Input	Input
MessageId	Input	Input	Input	Input	Input	Input	Input	Input
BusObjCacheSize	Input	Input	Input	Input	Input	Input	Input	Input
LastPage	-	Output	-	-	-	-	-	Output
NewQuery	-	Input	-	-	-	-	-	Input
PageSize	-	Input	-	-	-	-	-	Input
StartRowNum	-	Input	-	-	-	-	-	Input

Table 8. EAI Siebel Adapter Method Arguments

Argument	Query	QueryPage	Sync	Upsert	Update	Insert	Delete	Execute
ViewMode	Input	Input	Input	Input	Input	Input	Input	Input
SortSpec	-	Input	-	-	-	-	-	Input

Table 9 presents each argument of EAI Siebel Adapter methods.

Table 9. EAI Siebel Adapter Method Arguments

Argument	Display Name	Description
IntObjectName	Integration Object Name	The name of the integration object that is to be deleted.
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.
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 only use the user keys to perform query.
DeleteByUserKey	Delete By User Key	A Boolean argument. Forces the EAI Siebel Adapter to only use the user keys to identify a record.
ErrorOnNonExistingDelete	Error On Non Existing Delete	A Boolean argument. Determines whether or not the EAI Siebel Adapter should abort the operation if no match is found.
SiebelMessage	Siebel Message	The input or the output integration object instance.

Table 9. EAI Siebel Adapter Method Arguments

Argument	Display Name	Description
SearchSpec	Search Specification	This argument allows you to specify complex search specifications as free text in a single method argument. See “Search Specification” on page 134 for details.
StatusObject	Status Object	This argument tells EAI Siebel Adapter whether or not to return a status message.
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 process instance ID, then you can use this argument to specify the ID.
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.
LastPage	Last Page	Boolean indicating whether or not the last record in the query result set has been returned.
NewQuery	New Query	Default is False. Boolean indicating whether a new query should be executed. If set to True, a new query is executed flushing the cache for that particular integration object.
PageSize	Page Size	Default is 10. Indicates the maximum number of integration object instances to be returned.
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.

Table 9. EAI Siebel Adapter Method Arguments

Argument	Display Name	Description
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.
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 <i>Siebel Tools Reference</i> .

Query Method

You pass the Query method a Query By Example (QBE) integration object instance, a Primary Row Id, or a Search Specification. The adapter uses this input as criteria to query the base business object and to return a corresponding integration object instance. For example, to query Contact records with first name *David* you need to pass the following required input arguments to the Query method of EAI Siebel Adapter:

- SiebelMessage.IntObjName with value set to *Test Contact*
- SiebelMessage.ListOfTest Contact.Contact.First Name with value set to *David*

Now, if you need to further limit the output based on a value in the child component of the Test Contact (for example, to only query the Contact records with first name *David* and Action Type of *Call*), then you need the following required input arguments:

- SiebelMessage.ListOfTest Contact.Contact.First Name with Value set to *David*
- SiebelMessage.IntObjName with value set to *Test Contact*
- SiebelMessage.ListOfTest Contact.Contact.ListOfAction.Action.Type, with Value set to *Call*

Note that this still returns the contacts with the first name *David*, even if they do not have an activity of type *Call*, but it does not list their activities. For an example of using the search specification method argument to limit the scope of your query see [“Search Specification” on page 134](#).

NOTE: When using the EAI Siebel Adapter, to query all the business component records, you do not need to specify any value in the Object Id process property of the workflow process. In this case not specifying an ID works as a wildcard. If you want to query Siebel data using the EAI Siebel Adapter with the Query method and a property set containing a query by example search criteria, then all the fields that make up the user key for the underlying integration object component must exist in the property set. You can use an asterisk (*) as a wildcard for each one of the fields, but all of the user key fields must exist; otherwise, no record is returned.

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 [Table 9 on page 120](#). You can also use method arguments such as OutputIntObjectName, SearchSpec, SortSpec, ViewMode, and StartRowNum to dictate which records to be returned.

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. Please 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.

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, inserts, or deletes on business components. Some rules apply to the results of this method:

- If a child component is not present in the integration object instance, the corresponding business component rows are left untouched.
- If a child component is present in the integration object instance, but contains no instances so that there is only an empty container, then records in the corresponding business component are deleted.
- If a child component is present in the integration object instance, and contains some instances, the business component rows corresponding to the instances are updated or created and any business component row that does not have a corresponding integration component instance is deleted.
- The Sync method applies the operation sequentially to each root Integration Component (because each previous Integration Component is written to the database) but does not do this for any child Integration Component.

NOTE: The Synchronize method only updates the fields specified in the integration component instance.

Upsert Method

The Upsert method is similar to the Synchronize method with the following exceptions:

- The Upsert method does not delete any records.
- The Upsert method applies the operation sequentially to each root Integration Component (because each previous Integration Component is written to the database) but does not do this for any child Integration Component.

Insert Method

This method is also similar to the Synchronize method with the exception that the EAI Siebel Adapter errors out if a match 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 need to 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, 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.

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. For example, if you send an order with one order item to the EAI Siebel Adapter, it will take the following actions:

- 1** Queries for the order and if it finds a match, it updates the record.
- 2** Updates or inserts the new order item depending on if 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. A business component is deleted as specified by an integration object. If you specify any child integration component instances, then the fields of an integration component instance are used to query a business component.

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. EAI Siebel Adapter attempts to delete matching records in the business component before deleting the parent record.

Execute Method

The Execute method can be specified on 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 (default operation)
- upsert
- update
- updatesync
- insert
- insertsync
- delete
- 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 details, see [Table 11 on page 129](#).

Operations that include the word *sync* in the name cause deletion of unmatched child records, whereas update, insert, and upsert do not delete any children. [Table 10](#) presents the overview of the six related operations.

Table 10. EAI Siebel Adapter Execute Method Operations

EAI Siebel Adapter Action	upsert	sync	update	updatesync	insert	insertsync
Error on Match Found	No	No	No	No	Yes	Yes
Error on Match Not Found	No	No	Yes	Yes	No	No
Delete Unmatched Children	No	Yes	No	Yes	No	Yes

NOTE: You should use the Execute method when you need to mix different operations on different components within a single integration object; otherwise, you should use the other methods.

An XML document sent to a Siebel application can include operations that describe whether a particular data element needs to 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.

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, and none. If the operation is not specified on the root component, the sync operation is used as the default.

NOTE: Specifying operation within <ListOf> tag is not supported. For details on the <ListOf> tag, see *XML Reference: Siebel eBusiness Application Integration Volume V*.

Supported Operations for the Parent and Its Child Components

Table 11 presents the operation performed for a child component based on its parent component's operation and its own operation.

Table 11. Supported Operations

Parent Operation										
	query	query page	sync	upsert	update	update sync	insert	insert sync	delete	
Child Operation	query	query	query	upsert	upsert	update	update	insert	insert	delete
	query page	query	query	upsert	upsert	update	update	insert	insert	delete
	sync	query	query	sync	sync	sync	sync	sync	sync	delete
	upsert	query	query	upsert	upsert	upsert	upsert	upsert	upsert	delete
	update	query	query	update	update	update	update	upsert	upsert	delete
	update sync	query	query	update sync	update sync	update sync	update sync	sync	sync	delete
	insert	query	query	insert	insert	insert	insert	insert	insert	delete
	insert sync	query	query	insert sync	insert sync	insert sync	insert sync	insert sync	insert sync	delete
	delete	query	query	delete	delete	delete	delete	delete	delete	delete

XML Examples

The following XML example demonstrates using upsert and delete operation to delete a particular child 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>
```

The following example illustrates updating multiple corresponding top level parent business component records with one XML file.

```
<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>
```

MVGs in EAI Siebel Adapter

Multi-value 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 details on MVGs, see [Chapter 1, “About 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 CSSEAllIntCompFieldDef) 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 only when the Auto Primary property of the underlying MVLink is specified as Selected or None. If Auto Primary is defined as Default, then the Object Manager does not allow the EAI Siebel Adapter to set the primary. The exception to this rule are all the visibility MVG components (components whose records are used by Object Manager to determine who is going to see their parent records). For details on Auto Primary property, see *Siebel Tools Reference*.

Setting a Primary Address for an Account

You have an account with multiple shipping addresses in a Siebel application. None of these addresses are marked as the primary address for the account and you want to select one of them as the primary shipping address.

To specify an address as a primary

- 1 Create your XML file and insert `<IsPrimaryMVG= 'Y'>` before the address you want to identify as the primary address for the account as shown below:

```
<?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.

Search Specification

The SearchSpec input method argument is applicable to 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 that have a phone number.

NOTE: The AND operator is the only allowed operator among different integration components. You use 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. For example, using the same integration object definition as in previous examples, the `[Contact.PhoneNumber] IS NOT NULL` queries every account. Then for each account, it queries only contacts that have a phone number. If you want to query only accounts that are associated with contacts that have a phone number specified, then you need to create another business object, and an integration object based on that business object, which has contact as a root component, and account as its child component.

The following procedure illustrates how to use the SearchSpec to query specific accounts.

To query accounts and addresses based on integration object's SearchSpec field

- 1 From the application-level menu, choose View > Site Map > Business Process Administration > Workflow Processes.
- 2 Create a new workflow process based on the Sample Account business object.

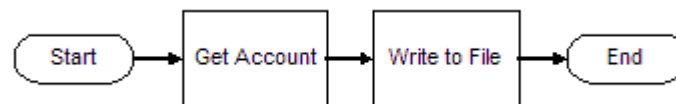
NOTE: Make sure all the fields you need are activated in the object.

- 3 Define the process properties.

Workflow process properties are global to the entire workflow. The Account Message is defined to identify the outbound Account as a hierarchical structure. The Error Message, Error Code, Object Id, and Siebel Operation Object Id properties are included in each workflow by default.

Name	Data Type	In/Out
Account Message	Hierarchy	In/Out
Error Code	String	In/Out
Error Message	String	In/Out
Object Id	String	In/Out
Process Instance Id	String	In/Out
Siebel Operation Object Id	String	In/Out

- 4 Click on the Process Designer tab in the bottom applet and design your workflow process as follows.



- 5** Double-click on the first step, after Start, and set it up to invoke the EAI Siebel Adapter to query the accounts and addresses for all records that match the desired search specification—for example, accounts created today with State equal to “IL.” To achieve this you need the following input and output arguments.

Input Arguments	Type	Value
Account Message	Literal	Sample Account
Search Specification	Expression	'[Account.Created] = ' + Today() + '[Account_BusinessAddress.State] = “IL”'

- 6** Double-click on the second step and set it up to write the record set to a text file using the EAI XML Write to File business service. Use the following arguments with the Write Siebel Message method.

Input Arguments	Type	Value	Property Name	Property Data Type
File Name	Literal	c:\acctnt&add.xml	-	-
Siebel Message	Process Property	-	Account Message	Hierarchy

The EAI XML Write to File business service converts the hierarchical message to XML and writes the result to the text file named in the File Name argument as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>

  <?Siebel-Property-Set EscapeNames="false"?>

- <SiebelMessage MessageId="1IS-7LT" IntObjectFormat="Siebel
Hierarchical" MessageType="Integration Object"
IntObjectName="Sample Account">

- <ListOfSampleAccount>

- <Account>
```



```
<Created>04/05/2002 07:41:10</Created>

<CSN>1IS-1DBRT</CSN>

<Location>Princeton</Location>

<Name>1st Account created today</Name>
- <ListOfAccount_BusinessAddress>
- <Account_BusinessAddress IsPrimaryMVG="N">

  <City>Abbott Park</City>

  <Country>USA</Country>

  <State>IL</State>

  <StreetAddress>1 Abbott Rd. D3m, B 3</StreetAddress>

  <AddressName>1 Abbott Rd. D3m, B 3, Abbott Park, IL</
AddressName>

  </Account_BusinessAddress>
</ListOfAccount_BusinessAddress>

</Account>
- <Account>

  <Created>04/05/2002 07:42:27</Created>

  <CSN>1IS-1DBRY</CSN>

  <Location>Orange</Location>

  <Name>2nd Account created today</Name>
- <ListOfAccount_BusinessAddress>
- <Account_BusinessAddress IsPrimaryMVG="Y">

  <City>Chicago</City>

  <Country>USA</Country>

  <State>IL</State>
```

```
<StreetAddress>1 BOP, 7th Floor</StreetAddress>  
  
<AddressName>1 BOP, 7th Floor, Chicago, IL</AddressName>  
  
</Account_BusinessAddress>  
  
</ListOfAccount_BusinessAddress>  
  
</Account>  
  
</ListOfSampleAccount>  
  
</SiebelMessage>
```

Language-Independent Code

If the user Property `AllLangIndependentVals` is set to `Y` at the integration object level, then 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 (M)LOVs increases the EAI Siebel Adapter CPU usage by about 5%, but allows easier communication between systems that operate on different languages.

LOV Translation

The Siebel application distinguishes two types of lists of values (LOV): multilingual LOV (MLOV) and single-language LOV.

Multilingual LOV (MLOV) stores a language-independent code (LIC) in the Siebel Database that gets translated to a language-dependent value (LDV) for active language by Object Manager. MLOVs are distinguished by having Translation Table specified on 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 should use LDV (N = no translation necessary) or LIC (Y = translation needed) for such LOVs. For details, see [Table 6 on page 67](#).

Translating to LIC impacts performance but allows easier cooperation between systems that operate on different languages. This option should be especially used by various import and export utilities. Default value is *undefined* for backward compatibility with 6.x release behavior.

[Table 12](#) explains the behavior of Siebel Adapter according to the integration object user property *AllLangIndependentVals* values.

Table 12. Siebel Adapter's Behavior for the User Property AllLangIndependentVals

AllLangIndependentVals	Y	N	Undefined
LOV	LIC	LDV	LDV
MLOV	LIC	LDV	LIC

EAI Siebel Adapter 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 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 should be 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 may be needed because a business component may be updating multiple tables, either as extension tables or through implicit or explicit joins.

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, 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 can not be found, 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, 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, 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, the EAI Siebel Adapter proceeds with the requested operation.

Modification IDs

To determine which Mod Id fields need to be used as part of a Modification Key, you expose Mod Id fields for tables whose columns may be updated by that integration object. In some situations you might need to add corresponding integration component fields as well as business component fields.

NOTE: EAI Siebel Adapter can update base and extension tables. It may even update joined table columns through picklists that allow updates.

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 need to make sure it is active if it is needed for your business processes.

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, you need to manually add it to the integration object definition by copying the Mod Id field and changing the properties.

Modification ID for a Joined Table

A joined table's Mod Id field needs to be manually added in both business component and integration object definitions. Business component Mod Id fields for joined tables should:

- 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

MVG and MVGAssociation Integration Components

For integration components that are of type MVG or MVGAssociation, in addition to the above steps, you need to 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 EAI Siebel Adapter 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 > 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 c](#) is the name of the extension table.
- 2** Create business component fields for Mod Ids of Joined Tables. For the above example, create a new field in 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 needs 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 compile a new .srf.
- 8** Modify client program to use the Modification Key mechanism.
 - a** The client program should store the value of the Modification IDs when it queries data from Siebel Database.
 - b** The client program should send exactly the same values of the Modification IDs that it retrieved from Siebel Database when sending an update.
 - c** The client program should not send in any Modification IDs when sending a new record to the Siebel application. If this is violated, 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 as 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.
- 1** Activate integration component field Mod Id.
 - a** Set Name, External Name, XML Tag properties to Mod Id
 - b** Set External Data Type property to DTYPE_NUMBER
 - c** Set External Length property to 30
 - d** Set Type property to System
- 2** Add integration component field S_ORG_EXT.Mod Id.
 - a** Set Name, External Name, XML Tag properties to S_ORG_EXT.Mod Id
 - b** Set External Data Type property to DTYPE_NUMBER
 - c** Set External Length property to 30
 - d** Set Type property to System
- 3** Add integration component field CX_Primary Organization-S_BU.Mod Id.
 - a** Set Name, External Name, XML Tag properties to CX_Primary Organization-S_BU.Mod Id
 - b** Set External Data Type property to DTYPE_NUMBER
 - c** Set External Length property to 30
 - d** 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.

- 1** For field Mod Id create a new user property with the name of AssocFieldName with a value of Mod Id.

- 2 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 the integration component example, Account (created in [“Integration Component Account_Organization Example”](#) on page 144) of Sample Account integration object, takes the following action:

- 1 Create a new Integration Component key called Modification Key.
- 2 Set the type of the key as Modification Key.
- 3 Add integration component fields Mod Id, S_ORG_EXT.Mod Id, and S_BU.Mod Id to the Modification Key.

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 gets triggered whenever a record is written. If you use both EAI Siebel Adapter to import data into Siebel application and run-time events, you should pay attention to the following:

For EAI Siebel Adapter, one call to EAI Siebel Adapter with an input message is a transaction. Within a transaction, EAI Siebel Adapter makes multiple Write Record calls. At any point in the transaction, if 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 invoked as soon as EAI Siebel Adapter makes Write Record calls even though EAI Siebel Adapter may be in the middle of a transaction. If you have export data workflows triggered on such events, this may lead to exporting data from Siebel applications that is not committed in Siebel applications and may get rolled back. It is also possible that your events get 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 EAI Siebel Adapter and run-time events use the business service EAI Transaction Service to figure out if a transaction (typically, EAI Siebel Adapter) is in progress. You may then want to skip processing that is not desirable when EAI Siebel Adapter is in progress.

For example, suppose you have a workflow to export Orders from Siebel applications that is triggered whenever the Order record is written. You also import Orders into Siebel applications using EAI. In such a situation, you do not want to export Orders while they are being imported because the import may get aborted and rolled back. You achieve this using the business service EAI Transaction 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 applications also support Workflow Policies as a triggering mechanism for workflows. You can use workflow policies instead of run-time events to avoid the situation discussed above. You should use Workflow Policies instead of Run-Time Events when possible.

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.

For example, if you are exchanging service requests with another application or partner, 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.

In order to use file attachments you first need to create Integration Objects. For details, see [Chapter 1, “About Integration Objects,”](#) and [Chapter 2, “Creating and Maintaining Integration Objects.”](#)

Siebel eAI offers the choice of integrating file attachments using MIME (the industry standard for exchanging multi-part messages), or including the attachment within the body of the XML document, referred to as an inline XML attachment. You should consider using inline XML attachments when integrating two instances of Siebel applications using file attachments.

Exchange of Attachments with External Applications

Siebel eAI supports bidirectional attachments 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 applications with other applications.

- **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. In this case, attachments are included within the body of the inline XML attachment. Inline XML attachments should be considered when integrating two instances of Siebel applications using file attachments. For details, see *XML Reference: Siebel eBusiness Application Integration Volume V*.

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.

The following checklist shows the high-level procedures you need to perform to use MIME to exchange attachments between Siebel applications and another external system.

Checklist

-
- ☐ Create an integration object using the EAI Siebel Wizard.
For details, see [“Creating the Integration Object” on page 149](#).
-
- ☐ Create an inbound or outbound Workflow process.
For details, see [“Creating Workflow Processes Examples” on page 150](#).
-
- ☐ Test your workflow process using Workflow Process Simulator.
For details, see [“The EAI MIME Hierarchy Converter” on page 156](#).
-

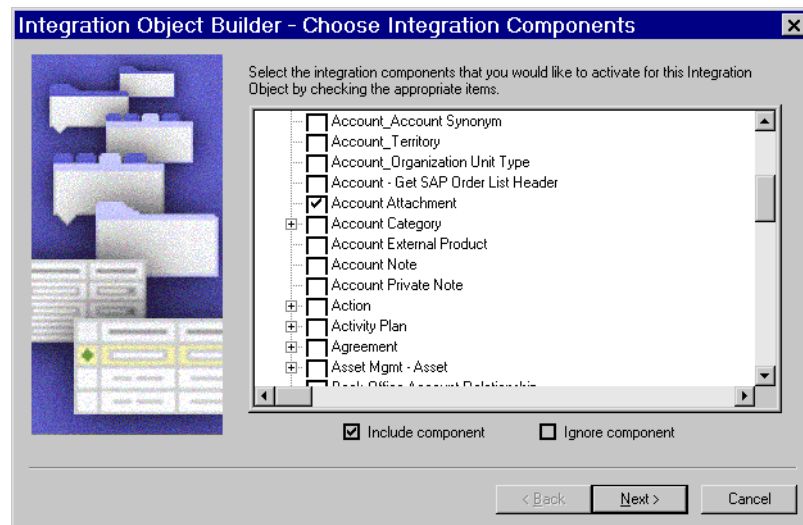
Creating the Integration Object

The following procedure guides you through the steps of creating an integration object.

To create a new Siebel integration object

- 1 Start Siebel Tools.
- 2 Create a new project and lock the project, or lock an existing project in which you want to create your integration object.
- 3 Choose File > New Object... to display the New Object Wizards.
- 4 Select the EAI tab, select the Integration Object icon, and click OK.

NOTE: When creating your integration object you need to select the Attachment integration object. The following figure illustrates this when the source object is Account.



- 5 Click Next to see a list of the warnings and errors generated by the Integration Object Builder.

- 6 Review and take necessary actions to address the issue.
- 7 Click Finish to complete the process of building the integration object.
- 8 In the Object Explorer, select Integration Object > Integration Component > Integration Component Field object.

The Integration Component and Integration Component Field applets appear.
- 9 Select the XXX_Attachment Component and the Attachment Id Component fields, and verify that the Data Type for the Attachment Id field is set to `DTYPE_ATTACHMENT`.
- 10 Compile the .srf file and copy it to the object directory under your Siebel Server directory as well as under your Tools directory.

NOTE: You need to stop the services before copying the .srf file. For details on the .srf file, see *Siebel Tools Reference*.

Creating Workflow Processes Examples

Depending on whether you are preparing for an outbound or an inbound attachment exchange, you need to design different workflow process as described in the following two procedures.

Outbound Workflow Process

To process the attachment for an outbound request you need to create a workflow process 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 process

- 1 Navigate to Workflow Process Designer.
- 2 Create a workflow process consisting of Start, End, and four Business Services. Set up each Business Service according to the task it needs to accomplish.

3 Define your process properties.

Set workflow process properties when you need a global property for the entire workflow.

Name	Data Type	Default String
SiebelMessage	Hierarchy	
Error Message	String	
Error Code	String	
Object Id	String	
Process Instance Id	String	
Siebel Operation Object Id	String	
MIMEHierarchy	Hierarchy	
SearchSpec	String	[Account.Name] = 'Sample Account'
MIMEMsg	String	

- 4 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	-	-
SearchSpec	Process Property	-	SearchSpec	String

Property Name	Type	Output Argument
SiebelMessage	Output Argument	Siebel Message

NOTE: For more information on using EAI Siebel Adapter, see [Chapter 5, “EAI Siebel Adapter.”](#)

- 5 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

Property Name	Type	Output Argument
MIMEHierarchy	Output Argument	MIME Hierarchy

NOTE: For more information on the EAI MIME Hierarchy Converter, see [“The EAI MIME Hierarchy Converter” on page 156](#).

- 6 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.

Input Argument	Type	Property Name	Property Data Type
MIME Hierarchy	Process Property	MIMEHierarchy	Hierarchy

Property Name	Type	Output Argument
MIMEMsg	Output Argument	MIME Message

NOTE: For more information on the EAI MIME Doc Converter, see [“The EAI MIME Doc Converter” on page 159](#).

- For the final step, you need to 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	-	MIMEMsg	String
File Name	Literal	c:\temp\account.txt	-	-

NOTE: For details on File Transport, see *Transports and Interfaces: Siebel eBusiness Application Integration Volume III*.

Inbound Workflow Process Example

To process the attachment for an inbound request, you need to create a workflow process to read the content from a file, convert the information into a Siebel Message, and send to EAI Siebel Adapter to update the database accordingly.

To create an inbound workflow process

- Navigate to Workflow Process Designer.
- Create a workflow process consisting of Start, End and four Business Services. Set up each Business Service according to the task it needs to accomplish.
- Define your process properties.

Set workflow process properties when you need a global property for the entire workflow.

- 4 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
MIMEMsg	Output Argument	Message Text

NOTE: For details on File Transport, see *Transports and Interfaces: Siebel eBusiness Application Integration Volume III*.

- 5 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	MIMEMsg	String

Property Name	Type	Output Argument
MIMEHierarchy	Output Argument	MIME Hierarchy

- 6** 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

- 7** 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

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. [Figure 27](#) illustrates the Siebel Message of a sample Account with attachments. This figure represents both input and output to the MIME Hierarchy Converter.

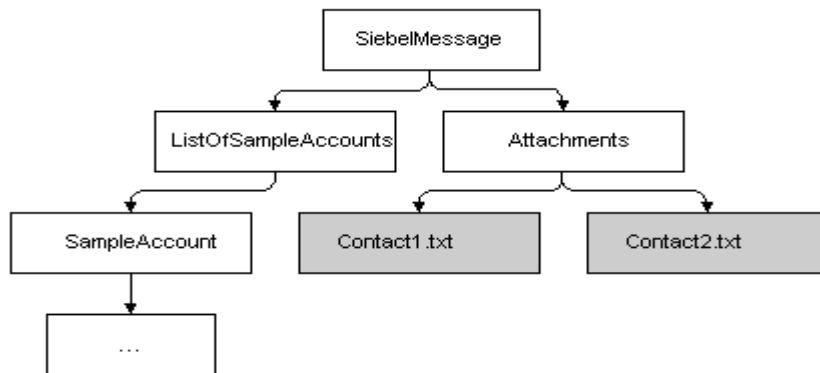


Figure 27. Sample Account with Attachments as Input to the MIME Hierarchy Converter

The output of this process is illustrated in [Figure 28](#).

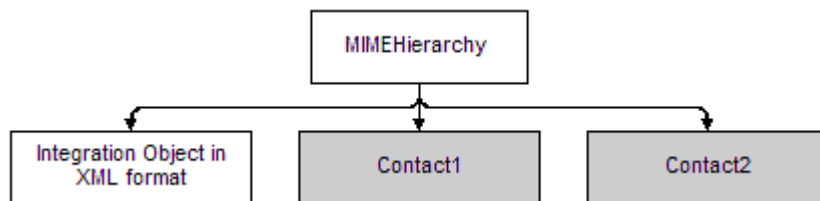


Figure 28. Output of a MIME Hierarchy Converter

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 under Attachments. In the event that there is no child of type Attachments in the Siebel Message, 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. [Figure 29](#) illustrates the incoming hierarchy.

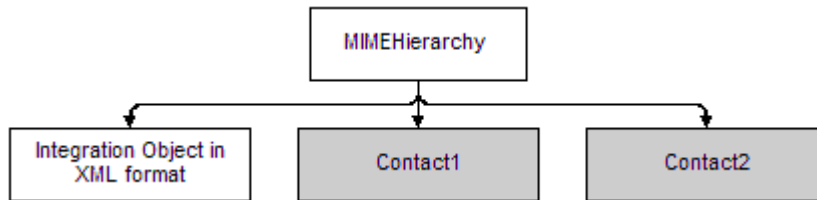


Figure 29. Output of a MIME Hierarchy Converter

The output of this process is illustrated in [Figure 27 on page 157](#). The output for this process is the same as the input.

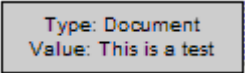
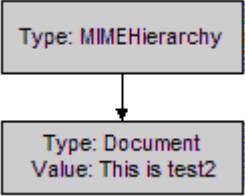
The EAI MIME Doc Converter

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.

Property	Description
MIME Hierarchy	Mapping to a MIME multi-part
Document	Mapping to MIME basic-part

Table 13 illustrates some examples of how a MIME Message maps to a MIME Hierarchy.

Table 13. Examples of MIME Message and MIME Hierarchy

MIME Message	MIME Hierarchy
MIME-Version: 1.0 Content-Type: application/xml Content-Transfer-Encoding: 7bit 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--	

EAI MIME Doc Converter Properties

The business service needs the following properties on the child property set as shown in [Table 14](#). These properties reflect the most accurate information on the data contained in the child property set.

Table 14. Properties for EAI MIME Doc Converter

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 message. When importing attachments, you should 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 as well as in the SiebelMessage. This property is automatically populated when you are exporting an attachment from 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, the Extension is used to retrieve the appropriate values from this property. If all three values are specified, the ContentType and ContentSubType values override the values retrieved from the Extension. If either the Extension or both ContentType and ContentSubType are not specified, the ContentType will be set to application and ContentSubType will have the value of octet-stream.

Table 14. Properties for EAI MIME Doc Converter

Property	Possible Values	Type	Description
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, the default value is used. The ContentType of multipart is used to represent file attachments in a MIME message. Other forms of values to describe a multipart is 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 the default value is used.

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 outputs 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.

This chapter describes the virtual business component (VBC), and its uses and restrictions. This chapter also describes how you can create a new VBC in Siebel Tools.

Overview of 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 and compile it into the siebel.srf file. The VBC calls a Siebel business service to provide a transport mechanism.

You can take two approaches to use virtual business components, as illustrated in Figure 30.

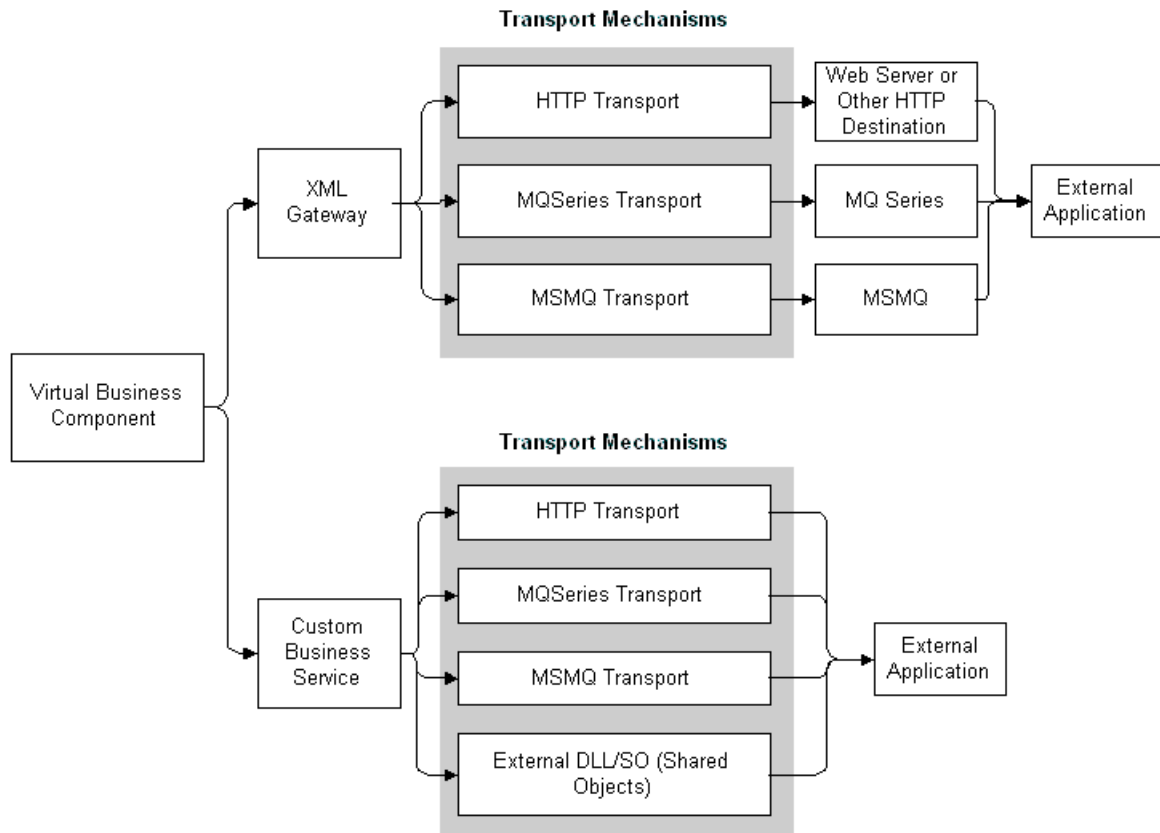


Figure 30. Two Approaches to Building Virtual Business Components

- Use the XML Gateway business service to pass data between the virtual business component and one of the Siebel transports, such as the EAI HTTP Transport, the EAI MQSeries AMI 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.

Enhancements to VBCs for This Version

The following new features and enhancements have been implemented in this version to enhance the functionality of the VBCs to better assist you in meeting your business requirements:

- Virtual business components (VBCs) support drill down from a VBC. You can drill down to a VBC from a standard BC, another VBC, or the same VBC.
- A parent applet can be based on a VBC.
- You can define virtual business components 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 XML format or property set by setting the VBC Compatibility Mode parameter to the appropriate version. For details, see [Table 15 on page 168](#).
- You can pass search and sort specifications to the business service used by a VBC.
- You can use Validation, Pre Default Value, Post Default Value, Link Specification, and No Copy attributes of VBC fields.
- You can use predefined queries with VBC.
- You can have picklists based on VBC and use the 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 No Insert, No Update, No Delete, Search Spec, Sort Spec, and Maximum Cursor Size business component properties.

Usage and Restrictions

- You can define a business object as containing both standard business components and virtual business components.

- When configuring applets based on VBCs, use CSSFrame (Form) and CSSFrameList (List) instead of specialized applet classes.
- Using the same name for the VBC field names and the remote data source field names may reduce the amount of required programming. (Optional)
- Virtual business components cannot be docked, so they do not apply to remote users.
- Virtual business components cannot contain a multi-value group (MVG).
- Virtual business components do not support many-to-many relationships.
- Virtual business components cannot be loaded using Enterprise Integration Manager.
- Standard business components can not contain multi-value group based on virtual business components.
- Virtual business components 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 virtual business components.
- You cannot use Workflow Monitor to monitor virtual business components.

Virtual Business Components

To use VBCs to share data with an external applications you need to perform the following high-level tasks:

Checklist

- ☐ Create a new Virtual Business Component.
For details, see [“Creating a New Virtual Business Component.”](#)
-

- ☐ Set the User Properties on Virtual Business Components (VBCs).
For details, see [“Setting User Properties for the Virtual Business Component” on page 168](#).
 - ☐ Configure your VBC Business Service:
 - Configure your XML Gateway Service or write your own Business Service.
For details, see [“XML Gateway Service” on page 169](#) and [“Custom Business Service Methods” on page 183](#).
 - Configure your external application.
For details, see [“External Application Setup” on page 183](#).
-

Creating a New Virtual Business Component

You create a new virtual business component in Siebel Tools.

To create a new virtual business component

- 1** Start Siebel Tools.
- 2** Lock the appropriate project.
- 3** Create a new record in the Business Component list applet in Siebel Tools.
- 4** Name the business component.
- 5** Select the project you locked in [Step 2](#).
- 6** Set the Class to the *CSSBCVExtern* class. This class provides the virtual business component functionality.

Setting User Properties for the Virtual Business Component

When defining the virtual business component, you must provide the user properties shown in [Table 15](#).

Table 15. Setting Virtual Business Component User Properties

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 only passes this user property as an input argument.
VBC Compatibility Mode	<p>(Optional) Determining 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 pre-7.5. Otherwise, a new property set and XML format will be passed.</p> <p>If you are creating a VBC in 7.5, there is no need to define this new user property since the default would be to use the new PropertySet from VBC and the new outgoing XML from the XML Gateway.</p> <p>For your existing VBC implementation you need to update your VBC definition by adding this new user property and setting it to Siebel xxx, where xxx is your desired version number.</p>

To define user properties

- 1 Start Siebel Tools.
- 2 Lock the appropriate project.
- 3 Click the Business Component folder in the Object Explorer to expand the hierarchical tree.

- 4 Select the business component you want to define user properties for.
- 5 Click the Business Component User Prop folder in the Object Explorer.
- 6 Choose Edit New Record to create a new blank user property record.
- 7 Type the name of the user property, such as Service Name, in the Name field.
- 8 Type the value of the user property, such as a business service name, in the Value field.
- 9 Repeat the process for every user property you want to define for this virtual business component.

NOTE: For list of different property sets and their format, see [“Examples of Outgoing XML Format” on page 174](#) and [“Examples of Incoming XML Format” on page 180](#).

XML Gateway Service

The XML Gateway business service communicates between Siebel applications and external data sources using XML as the data format. For details on XML format, see [“Examples of Outgoing XML Format” on page 174](#) and [“Examples of Incoming XML Format” on page 180](#). The XML Gateway business service can be configured to use one of the following transports:

- EAI MQSeries AMI Server Transport
- 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 virtual business component as shown in [Table 16](#). When using the XML Gateway, you need to specify the following user properties for your virtual business component.

Table 16. User Properties

Name	Value
Service Name	XML Gateway
Service Parameters	<i>variable1 name=variable1 value; variable2 name=variable2 value>;...</i>
Remote Source	<i>External Data Source</i>
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 should 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, you may use something like the following which is also illustrated in [Figure 31](#):

```
"Transport=EAI HTTP Transport;HTTPRequestURLTemplate=<your URL>;HTTPRequestMethod=POST"
```

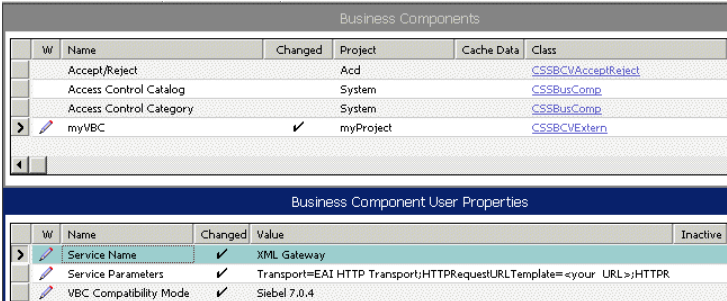


Figure 31. Setting Virtual Business Component User Properties

or if you want to specify the EAI MQSeries AMI Transport, you may use something like:

```
"Transport=EAI MQSeries AMI Transport;MqPolicyName=<policy name>;MqSenderServiceName=<sender service name>;MqModelQueueName=<queue name>;MqPhysicalQueueName=<p queue name>;..."
```

You can also implement VBC with MQSeries. The following procedure lists the steps you need to take to implement this.

To implement VBC with MQSeries

- 1 Call the EAI Business Integration Manager (Server Request) business service.

NOTE: You do not need to define the EAI MQSeries Server Transport business service as the transport on the service parameters line. MQSeries is usually installed on the same machine as the Siebel Server and not installed on the client machine; therefore, references to the EAI MQSeries Server Transport as the transport parameter for the VBC will not work.

- 2 Define another service parameter for the name of a workflow process 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 process, EAITEST, to call the EAI MQSeries Server Transport with the SendReceive method.
 - 4** Define a new process property, < Value > , on the workflow process and use it as an output argument on the EAI MQSeries Server Transport step in the workflow process.

XML Gateway Methods

The XML Gateway provides the methods presented in [Table 17](#).

Table 17. XML Gateway Methods

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 [Table 18](#).

Table 18. XML Gateway Arguments

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, a user property Remote Source can be defined to allow the Transport Services to determine the remote destination and any connect information. If this user property is defined, it is passed to every request as the <remote-source> tag.
Business Component Id	Unique key for the given business component.
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.

Examples of Outgoing XML Format

Examples of the XML documents generated and sent by the XML Gateway to the external system are presented in [Table 19](#). These examples are based on the example in “[Custom Business Service Example](#)” on [page 203](#). See [Appendix C, “DTDs for XML Gateway Business Service,”](#) for examples of the DTDs that correspond to each of these methods.

NOTE: The XML examples provided in this chapter have extraneous carriage returns and line feeds for ease of reading. Please delete all the carriage returns and line feeds before using any of the examples.

Table 19. Outgoing XML Tags and Descriptions

Method	Format of the XML Stream	Description
Delete Request	<pre> <siebel-xmlxt-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-xmlxt-delete-req> </pre>	<p>siebel-xmlxt-delete-req. This tag requests removal of a single record in the remote system.</p>
Init Request	<pre> <siebel-xmlxt-fields-req> <buscomp id="1">Contact</buscomp> <remote-source>http://throth/servlet/ VBCContacts</remote-source> </siebel-xmlxt-fields-req> </pre>	<p>siebel-xmlxt-fields-req. This tag fetches the list of fields supported by this instance.</p> <p>buscomp Id. The business component ID.</p> <p>remote-source. The remote source from which the service is to retrieve data for the business component.</p>

Table 19. Outgoing XML Tags and Descriptions

Method	Format of the XML Stream	Description
Insert Request	<pre> <siebel-xmltext-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-xmltext-insert-req> </pre>	<p>siebel-xmltext-insert-req. This tag requests the commit of a new record in the remote system.</p> <p>The insert-req XML stream contains values for fields entered through the business component.</p>
PreInsert Request	<pre> <siebel-xmltext-preinsert-req> <buscomp id="1">Contact</buscomp> <remote-source>http://throth/ servlet/VBCContacts</remote-source> </siebel-xmltext-preinsert-req> </pre>	<p>siebel-xmltext-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 BusComp interface.</p>

Table 19. Outgoing XML Tags and Descriptions

Method	Format of the XML Stream	Description
Query Request	<pre> <siebel-xmlxt-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-xmlxt-query-req> </pre>	<p>siebel-xmlxt-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.</p> <p>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.</p> <p>search-string. The search specification used to query and filter the information.</p> <p>search-spec. Hierarchical representation of the search-string. For details, see “Search-Spec Node-Type Types” on page 179.</p> <p>sort-spec. List of sort fields and sort order.</p>

Table 19. Outgoing XML Tags and Descriptions

Method	Format of the XML Stream	Description
Update Request	<pre><siebel-xmlxt-update-req> <buscomp id="2">Contact</buscomp> <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-xmlxt-update-req></pre>	<p>siebel-xmlxt-Update-req. This tag requests changes to the field values for an existing row.</p> <p>All values for the record are passed in with <value> tags, with the changed attribute identifying the ones that have been changed through the Siebel application.</p>

Search-Spec Node-Type Types

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. [Table 20](#) shows the different search-spec node-types and their values.

Table 20. Search-Spec Node-Types

node-type	PropertySet/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 Incoming XML Format

Table 21 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 example in “Custom Business Service Example” on page 203. See Appendix C, “DTDs for XML Gateway Business Service,” for examples of the DTDs that correspond to each of these methods.

Table 21. Incoming XML Tags and Descriptions

Method	Format of the XML Stream	Description
Delete Return	<siebel-xmltext-delete-ret />	siebel-xmltext-delete-ret. Only the XML stream tag is returned.
Error	<siebel-xmltext-status> <status-code>4</code> <error-field>Name</error-field> <error-text>Name must not be empty</error-text> </siebel-xmltext-status>	<p>Format of the XML stream expected by the Siebel application in case of an error in the external application. The tags for this XML stream, including the entire XML stream, are optional. If the error is specific to a field, the field name should be specified.</p> <p>siebel-xmltext-status. This tag is used to check the status returned by the external system.</p> <p>status-code. This tag overrides the return value.</p> <p>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 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.”</p>

Table 21. Incoming XML Tags and Descriptions

Method	Format of the XML Stream	Description
Init Return	<pre><siebel-xmlxt-fields-ret> <support field="AccountId" /> <support field="Name" /> <support field="Phone" /> <support field="Location" /> <support field="AccessId" /> </siebel-xmlxt-fields-ret></pre>	<p>siebel-xmlxt-fields-ret. The fields-ret XML stream return contains the list of VBC fields supported by the external application for this instance.</p> <p>The following field names are reserved by the Siebel application and should not appear in this list:</p> <p>Id, Created, Created By, Updated, Updated By.</p>
Insert Return	<pre><siebel-xmlxt-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-xmlxt-insert-ret></pre>	<p>siebel-xmlxt-insert-ret. If the remote system has inserted records, 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.</p>
PreInsert Return	<pre><siebel-xmlxt-preinsert-ret> <row> <value field="Location">San Jose</ value> </row> </siebel-xmlxt-preinsert-ret></pre>	<p>siebel-xmlxt-preinsert-ret. Returns default values for each field, if there is any default value.</p>

Table 21. Incoming XML Tags and Descriptions

Method	Format of the XML Stream	Description
Query Return	<pre><siebel-xmlxt-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> </row> </siebel-xmlxt-query-ret></pre>	<p>siebel-xmlxt-query-ret. The query-ret XML stream contains the result set that matches the criteria of the query.</p> <p>row. This tag indicates the number of rows returned by query. Each row should contain one or more < values > . The attributes which appear in < row > tags must be able to uniquely identify rows. If there is a unique key in the remote data source, it should appear in the result set. If not, a unique key should be generated. It is necessary to identify specific rows for DML operations.</p> <p>value. This tag specifies the field and value pairs and should be the same for each row in the set.</p>

Table 21. Incoming XML Tags and Descriptions

Method	Format of the XML Stream	Description
Update Return	<pre><siebel-xmlret-update-ret> <row> <value field="Location">San Jose</ value> <value field="Phone">(408)234-1029</ value> </row> </siebel-xmlret-update-ret></pre>	siebel-xmlret-update-ret. If the remote system updated fields, they 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

Once you have your XML Gateway Service configured, you need to set up your external application accordingly to be able to receive and respond to the requests. At a minimum, the external application needs to support the Init() and Query() methods, and depending upon the functionality provided by the VBC, the remaining methods may or may not be necessary.

Custom Business Service Methods

Your business service must implement the Init and Query methods as described in this section. The Delete, PreInsert, Insert, and Update methods are optional, and dependent upon the functionality required by the Virtual Business Component.

NOTE: Custom business services can be only based on the CSSService class, as specified in Siebel Tools.

These methods pass property sets between the virtual business component and the business service. Virtual business component 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 virtual business component.

When you are building a custom business service to allow virtual business component functionality with Siebel VB or Siebel eScript you can use one of the following methods to connect to an external database in the Service code:

- **Siebel VB Only.** Use the SQL functions using ODBC.
- **Siebel eScript Only.** Call out to a CORBA interface using the CORBACreateObject function.
- **Siebel VB or eScript.** Use a COM connection through the CreateObject or COMCreateObject functions to call an API supported by your RDBMS vendor or to call a COM object such as ActiveX DLL.

You may also choose to use the XML Gateway service to allow the connection for your VBC. For details, see [“XML Gateway Service” on page 169](#).

NOTE: For more information about property sets, programming in Siebel eScript, and programming in Siebel VB, see *Siebel Tools Reference* and *Siebel Tools Online Help*.

Common Method Parameters

[Table 22](#) shows the input parameters common to every method. Please note that all these parameters are at the root property set.

Table 22. Common Input Parameters

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 details, see Table 15 on page 168 .
Business Component Name	Name of the active virtual business component.

Table 22. Common Input Parameters

Parameter	Description
Business Component Id	Internally generated unique value that represents the virtual business component.
Parameters	Optional. The VBC's Service Parameters user property, if defined. For details, see Table 15 on page 168 . 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 details, see Table 15 on page 168 .

Once a response has been received, the method packages the response from the external data source into the outputs property set.

Business Services Methods and Their Property Sets

The following examples display each method's input and output property sets for a virtual business component Contact that displays simple contact information for a given account. These examples are based on the example in the [“Custom Business Service Example” on page 203](#).

NOTE: All the optional parameters have been omitted from these example to simplify them.

Delete The Delete method is called when a record is deleted. [Figure 32](#) illustrates the property set for the Delete input and is followed by its XML representation.

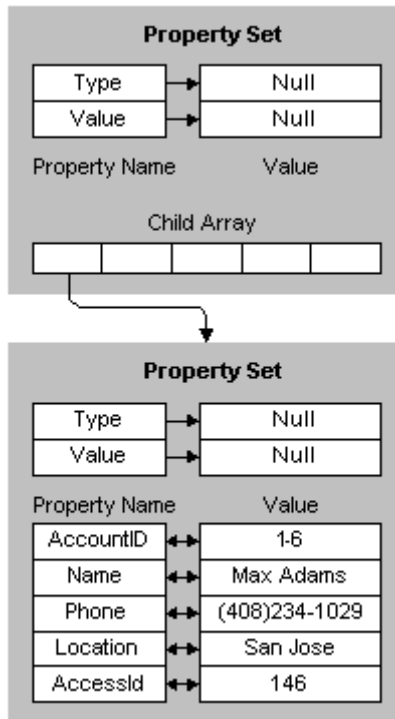


Figure 32. Delete Input Property Set

```
<?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>

```

Figure 33 illustrates the property set for Delete output and is followed by its XML representation.

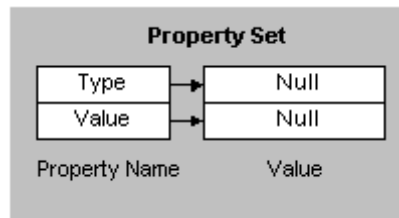


Figure 33. Delete Output Property Set

```

<?xml version="1.0" encoding="UTF-8" ?>

<?Siebel-Property-Set EscapeNames="true"?>

<PropertySet />

```

Error Return [Figure 32](#) illustrates the property set for the Error Return, when an error is detected. The illustration is followed by its XML representation.

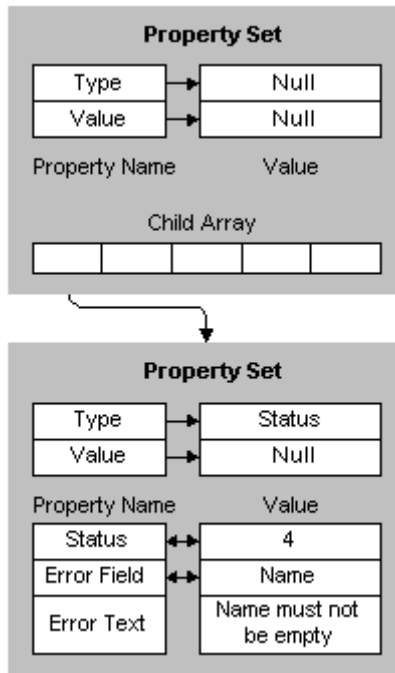


Figure 34. Error Return Property Set

```
<?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 The Init method is called when the virtual business component is first instantiated. It initializes the virtual business component. It expects to receive the list of fields supported by the external system. [Figure 35](#) illustrates the property set for Init input and is followed by its XML representation.

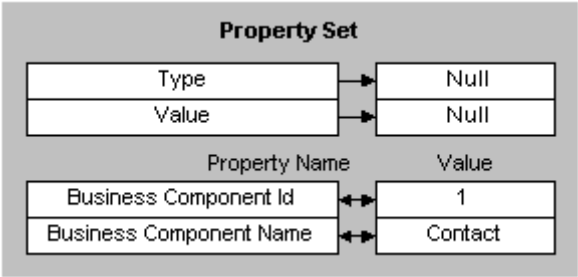


Figure 35. Init Input Property Set

```
<?xml version="1.0" encoding="UTF-8"?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet
  Business_spcComponent_spcId="1"
  Business_spcComponent_spcName="Contact" />
```

Figure 36 illustrates the property set for Init output and is followed by its XML representation.

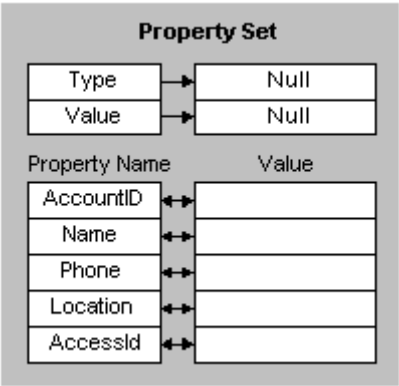


Figure 36. Init Output Property Set

```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet
  AccountId=" "
  Name=" "
  Phone=" "
  Location=" "
  AccessId=" " />
```

Insert The Insert method is called when a New Record is committed. [Figure 37](#) illustrates the property set for Insert input and is followed by its XML representation.

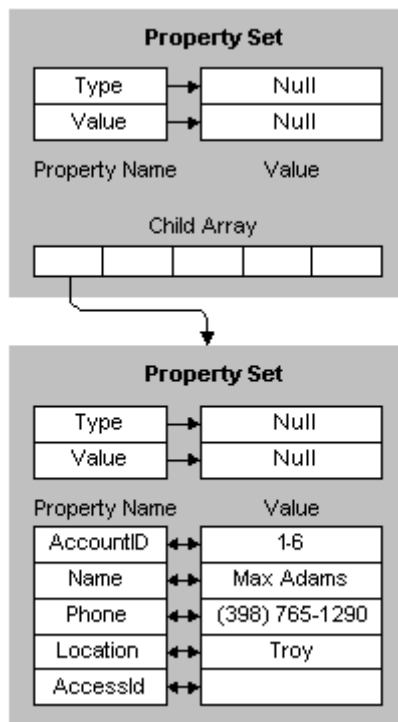


Figure 37. Insert Input Property Set

```
<?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>
```

Figure 38 illustrates the property set for Insert output and is followed by its XML representation.

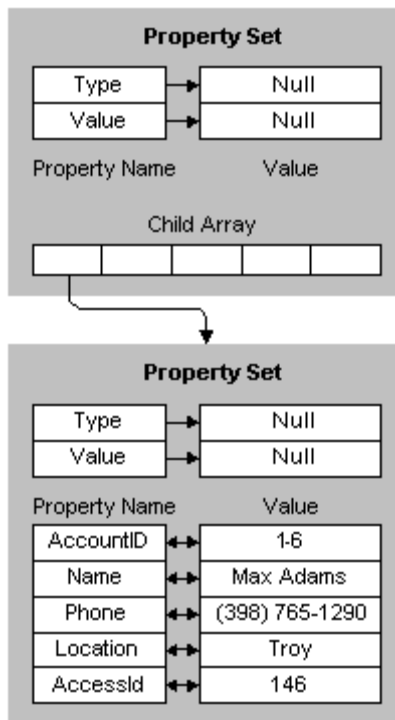


Figure 38. Insert Output Property Set

```
<?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 The PreInsert method is called when a New Record operation is performed. It supplies default values. [Figure 39](#) illustrates the property set for PreInsert input and is followed by its XML representation.

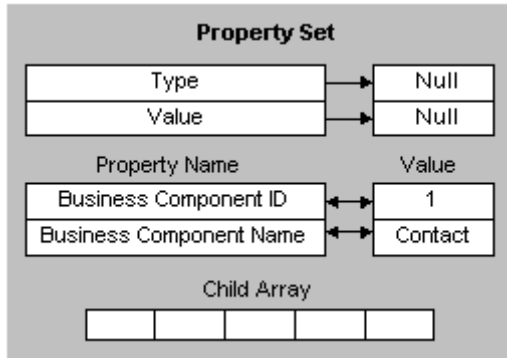


Figure 39. PreInsert Input Property Set

```
<?xml version="1.0" encoding="UTF-8"?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet
  Business_spcComponent_spcId="1"
```

```
Business_spcComponent_spcName="Contact" />
```

Figure 40 illustrates the property set for PreInsert output and is followed by its XML representation.

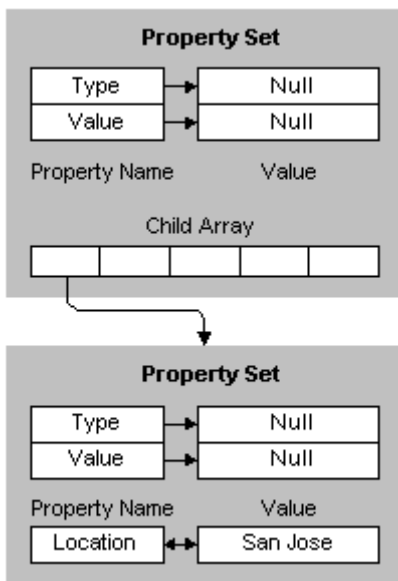


Figure 40. PreInsert Output Property Set

```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet>
  <PropertySet Location="San Jose" />
</PropertySet>
```

Query The Query method is called when a search is performed. The Query method must be supported by every virtual business component. Each record that matches the query is represented as a property set. For example, if 5 records match the query, there will be 5 child property sets. Each property set will contain a list of field names—field value pairs representing the values of each field for that particular record. [Figure 42](#) illustrates the property set for Query input and is followed by its XML representation.

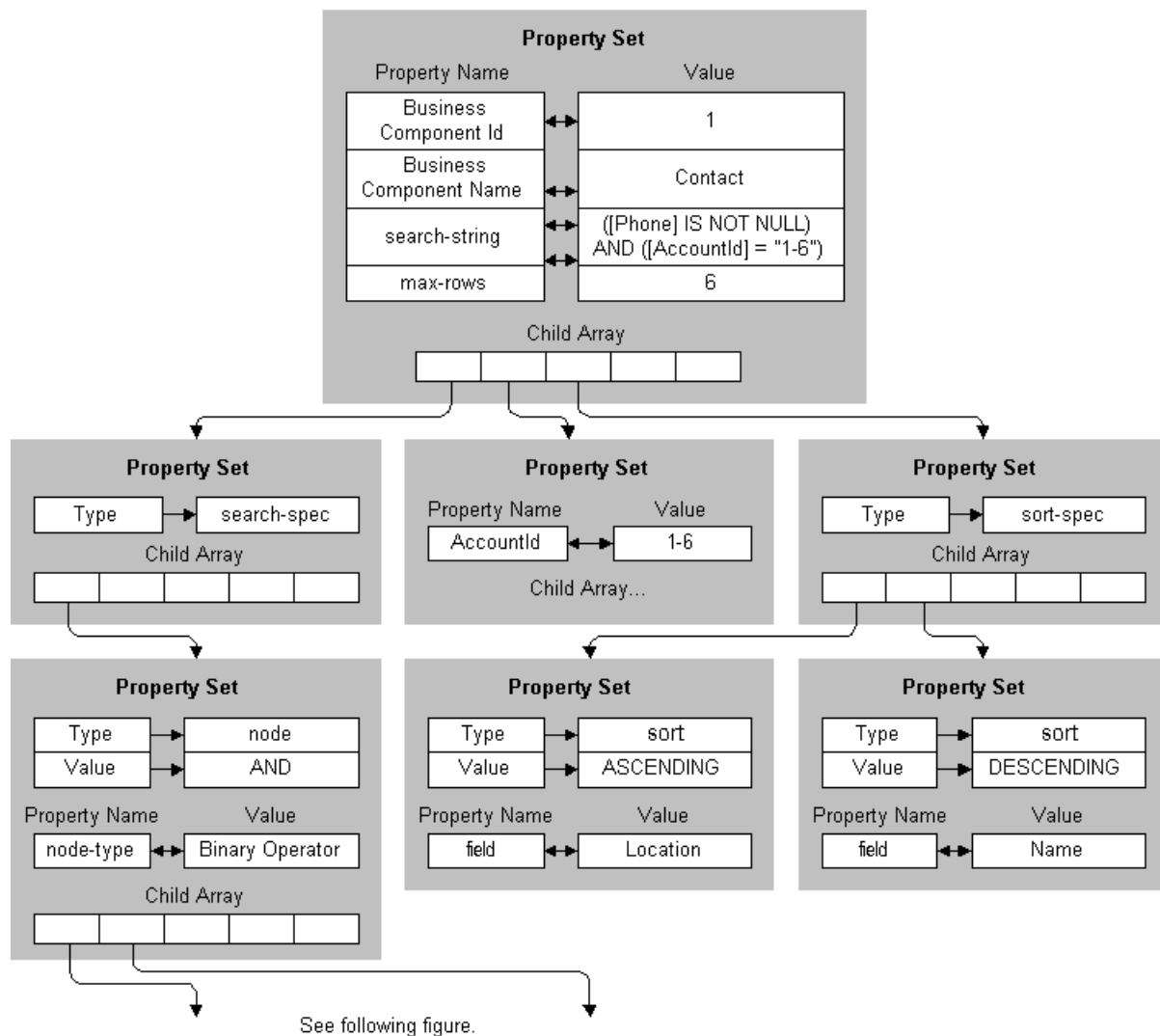


Figure 41. Query Input Property Set (Part 1)

From previous figure.

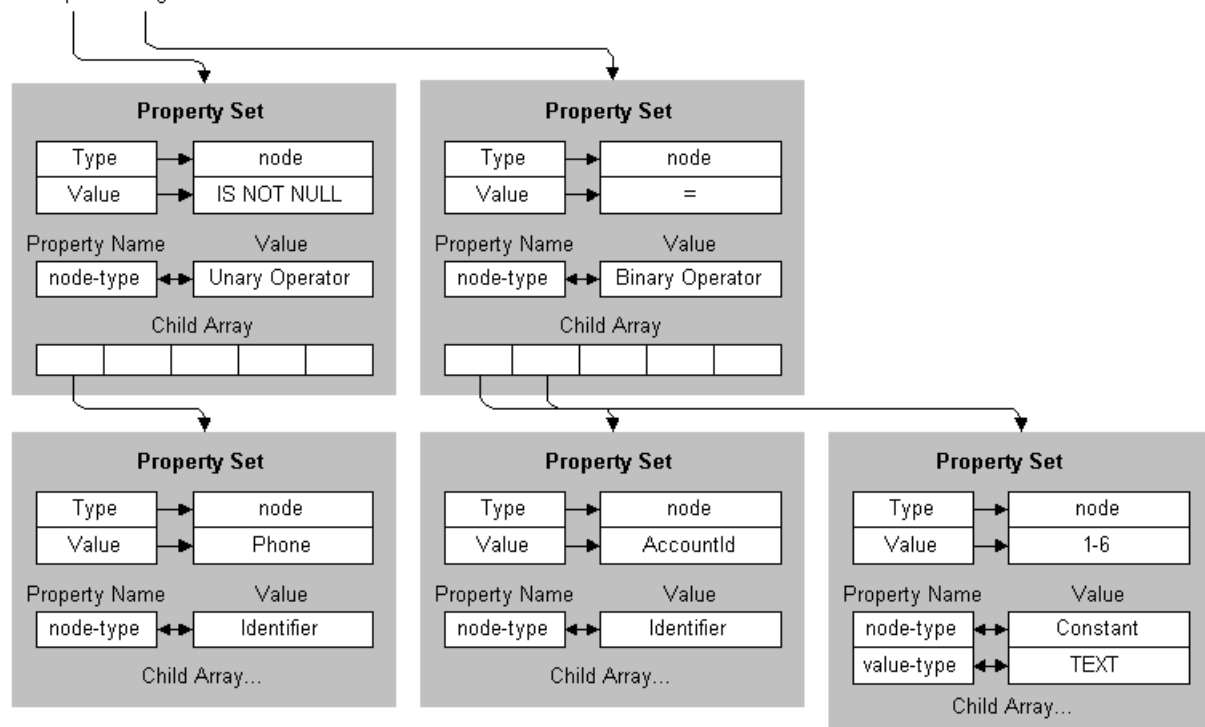


Figure 42. Query Input Property Set (Part 2)

```
<?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>
```

Figure 43 illustrates the property set for Query output and is followed by its XML representation.

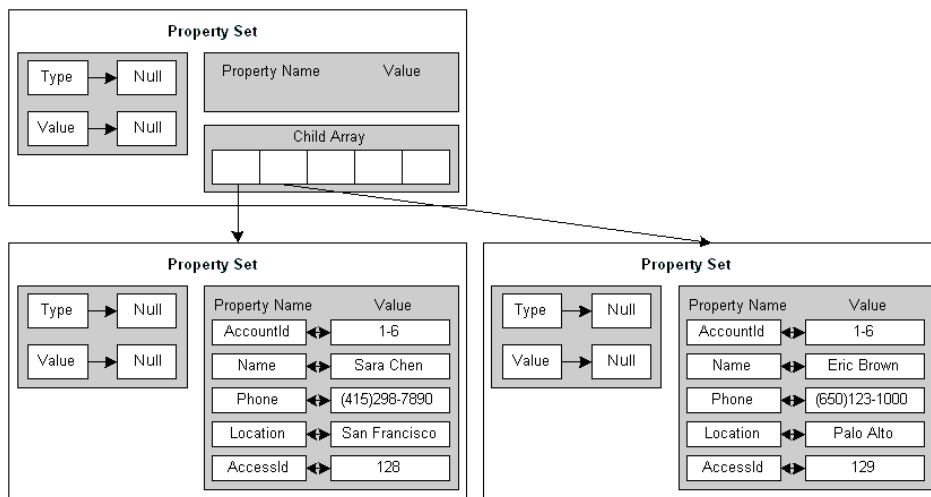


Figure 43. Query Output Property Set

```
<?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 The Update method is called when a record is modified. [Figure 44](#) illustrates the property set for Update input and is followed by its XML representation.

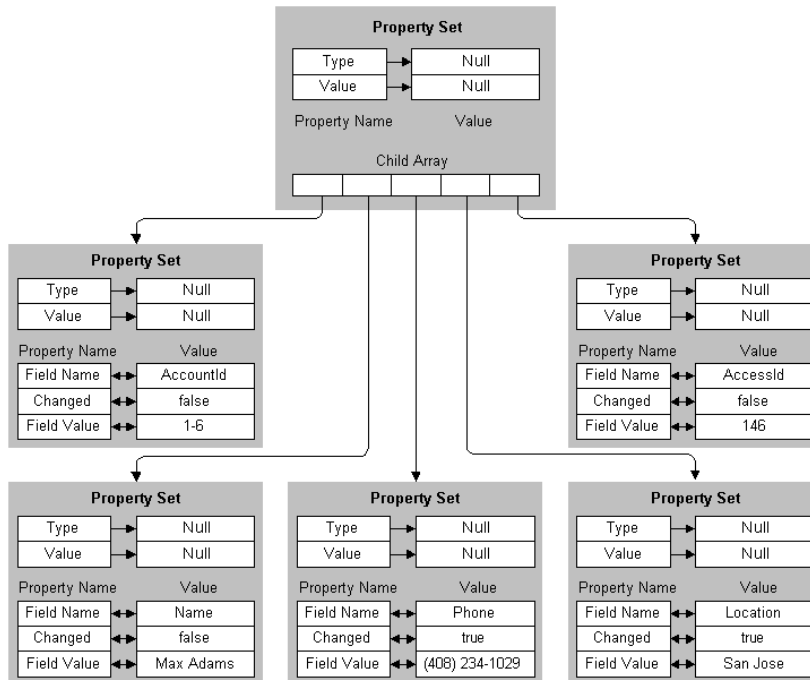


Figure 44. Update Input Property Set

```

<?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="Max Adams" />
  <PropertySet
    Field_spcName="Phone"
    Changed="true"
    Field_spcValue="(408)234-1029" />
  <PropertySet
    Field_spcName="Location"
    Changed="true"
    Field_spcValue="San Jose" />
  <PropertySet
    Field_spcName="AccessId"
    Changed="false"
    Field_spcValue="146" />
</PropertySet>
```

Figure 45 illustrates the property set for the Update output and is followed by its XML representation.

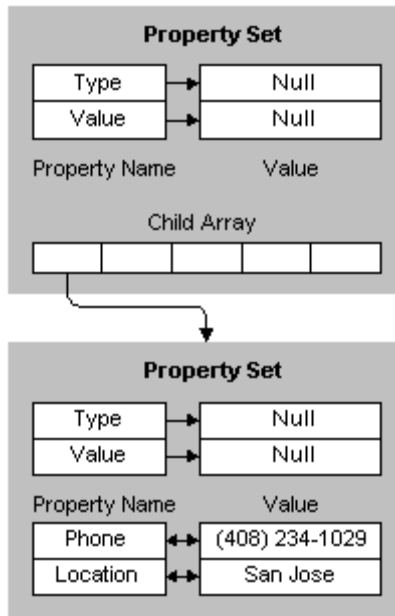


Figure 45. Update Output Property Set

```
<?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 Example

The following is an example of Siebel eScript implementation of a business service for a virtual business component. The fields configured for this simple virtual business component are AccountId, Name, Phone, Location, and AccessId. AccessId is the primary key in the external data source. AccessId is included in the virtual business component fields to make update and delete simple and is configured as a hidden field.

```
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 purpose...

    logPropSet(Inputs, "InitInputs.xml");

    Outputs.SetProperty("AccountId", "");

    Outputs.SetProperty("Name", "");

    Outputs.SetProperty("Phone", "");

    Outputs.SetProperty("AccessId", "");

    Outputs.SetProperty("Location", "");

    // For debugging purpose...

    logPropSet(Outputs, "InitOutputs.xml");

    return (CancelOperation);

}

function Query(Inputs, Outputs)
{
    // For debugging purpose...

    logPropSet(Inputs, "QueryInputs.xml");

    var selectStmt = "select * from Contacts ";

    var whereClause = " where ";

    var orderByClause = " order by ";

    // 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");

// convert the search-string into a where clause

searchString = stringReplace(searchString, '*', '%');
searchString = stringReplace(searchString, '[', ' ');
searchString = stringReplace(searchString, '|', ' ');
searchString = stringReplace(searchString, '~', ' ');
searchString = stringReplace(searchString, '"', "");

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
        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're only going to return no more than maxRows of records.

var count = rs.RecordCount();

if (maxRows != "")

    if (count > maxRows)

        count = maxRows

// We'll go through the recordset 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 purpose...
    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 purpose...

    logPropSet(Inputs, "PreInsertInputs.xml");

    var defaults = TheApplication().NewPropSet();

    defaults.SetProperty("Location", "KO");

    Outputs.AddChild(defaults);

    // For debugging purpose...

    logPropSet(Outputs, "PreInsertOutputs.xml");

    // clean up

    defaults = null;
```



```
        return (CancelOperation);
    }

    function Insert (Inputs, Outputs)
    {
        // For debugging purpose...

        logPropSet(Inputs, "InsertInputs.xml");

        var fieldList = "";

        var valueList = "";

        // Inputs should 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 need to 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().NewPropSet();

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");
```

```
// clean up

child = null;

row = null;

rs.Close();

rs = null;

conn.Close();

conn = null;

return (CancelOperation);

}

function Update (Inputs, Outputs)
{
    // For debugging purpose...

    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
    // 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 need 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're expecting 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().NewPropSet();

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, "UpdateOutputs.xml");

// clean up

child = null;

row = null;

rs.Close();

rs = null;

conn.Close();

conn = null;
```

```
        return (CancelOperation);
    }

    function Delete (Inputs, Outputs)
    {
        // For debugging purpose...

        logPropSet(Inputs, "DeleteInputs.xml");

        // Inputs should have only 1 child property set.

        var child = Inputs.GetChild(0);

        // In this example, we're only using the AccessId

        // (it's the primary key in the Contacts db)

        // for delete statement for simplicity.

        var deleteStmt = "delete from Contacts where AccessId = " +
            child.GetProperty("AccessId");

        // Now, let's delete the record from the data source.

        var conn = getConnection();

        conn.Execute(deleteStmt);

        // For debugging purpose...

        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().NewPropSet();

    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;
}
```

NOTE: For more examples of VBCs, see *Developing and Deploying Siebel eBusiness Applications*.

Predefined EAI Business Services

A

Siebel eBusiness Applications provide 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.

NOTE: For general information on using business services, refer to [Chapter 3, “Business Services.”](#)

Predefined EAI Business Services

[Table 23](#) presents the predefined Siebel eAI business services.

Table 23. Predefined EAI Business Services

Business Service	Class	Description
EAI XSD Wizard		Used to create integration objects based on XSD files.
EAI XML XSD Generator		Used to generate an XSD file from an integration object.
EAI Database Adapter		Used to interact with databases directly using SQL based on integration object definitions.
EAI Transaction Service	CSSBeginEndTransactionService	EAI Transaction service for working with Siebel transactions such as begin, end, or find out whether in transaction.
Workflow Process Manager (Server Request)	CSSSrmService	Submits workflow requests to a workflow process manager server component (WfProcMgr).

Table 23. Predefined EAI Business Services

Business Service	Class	Description
EAI MSMQ Transport	CSSMsmqTransService	EAI MSMQ Transport.
EAI MQSeries Server Transport	CSSMqSrvTransService	EAI MQSeries Server Transport.
EAI MQSeries AMI Transport	CSSMqAmiTransService	EAI MQSeries AMI Transport. For details, see <i>Transports and Interfaces: Siebel eBusiness Application Integration Volume III</i> .
EAI HTTP Transport	CSSHTTPTransService	EAI HTTP Outbound Transport. For details, see <i>Transports and Interfaces: Siebel eBusiness Application Integration Volume III</i> .
EAI Utility Service	CSSEAIUtilService	EAI Utility Service.
Oracle Adapter	CSSEAISqlAdapterService	EAI SQL Adapter. For details, see <i>Siebel eBusiness Connector for Oracle Guide</i> .
EAI Siebel Adapter	CSSEAISiebelAdapterService	EAI Siebel Adapter. For details, see Chapter 5, “EAI Siebel Adapter.”
EAI Query Spec Service	CSSEAIQuerySpecService	Used internally by EAI Siebel Adapter to convert SearchSpec method argument as string to an Integration Object Instance that EAI Siebel Adapter can use as a Query By Example object.
Oracle Procedure Adapter	CSSEAIODBCService	ODBC Service for Oracle connector. For details, see <i>Siebel eBusiness Connector for Oracle Guide</i> .
EAI Import Export	CSSEAIImportExportService	EAI Import Export Service (import and export integration object from or to XML).
EAI BTS COM Transport	CSSEAIBtsComService	EAI Siebel to BTS COM Transport.
EAI DLL Transport	CSSDIITransService	EAI DLL Transport. For details, see <i>Transports and Interfaces: Siebel eBusiness Application Integration Volume III</i> .

Table 23. Predefined EAI Business Services

Business Service	Class	Description
EAI Data Mapping Engine	CSSEDataTransformationEngine	EAI Data Transformation Engine. For details, see <i>Business Processes and Rules: Siebel eBusiness Application Integration Volume IV</i> .
No Envelope	CSSEAINullEnvelopeService	EAI Null Envelope Service. For details, see <i>XML Reference: Siebel eBusiness Application Integration Volume V</i> .
Siebel Message Envelope	CSSEAISMEnvelopeService	EAI Siebel Message Envelope Service. For details, see <i>XML Reference: Siebel eBusiness Application Integration Volume V</i> .
EAI Dispatch Service	CSSEAIDispatchService	Dispatch Service. For details, see <i>Business Processes and Rules: Siebel eBusiness Application Integration Volume IV</i> .
EAI Integration Object to XML Hierarchy Converter	CSSEAIIntObjHierCnvService	EAI Integration Object Hierarchy (also known as SiebelMessage) to XML hierarchy converter service. For details, see <i>XML Reference: Siebel eBusiness Application Integration Volume V</i> .
EAI MIME Hierarchy Converter	CSSEAIMimePropSetService	EAI MIME Hierarchy Conversion Service. For details, see Chapter 6, “Siebel eAI and File Attachments.”
EAI MIME Doc Converter	CSSEAIMimeService	MIME Document Conversion Service. For details, see Chapter 6, “Siebel eAI and File Attachments.”
EAI XML Converter	CSSEAIXMLCnvService	Converts between XML and EAI Messages. For details, see <i>XML Reference: Siebel eBusiness Application Integration Volume V</i> .
EAI XML Write to File	CSSEAIXMLPrtService	Print a property set to a file as XML. For details, see <i>XML Reference: Siebel eBusiness Application Integration Volume V</i> .

Table 23. Predefined EAI Business Services

Business Service	Class	Description
EAI XML Read from File	CSSEAIXMLPrtService	Read an XML file and parse to a property set. For details, see <i>XML Reference: Siebel eBusiness Application Integration Volume V</i> .
XML Converter	CSSXMLCnvService	Converts between XML documents and arbitrary Property Sets. For details, see <i>XML Reference: Siebel eBusiness Application Integration Volume V</i> .
XML Hierarchy Converter	CSSXMLCnvService	Converts between XML documents and XML Property Set or Arbitrary Property Set. For details, see <i>XML Reference: Siebel eBusiness Application Integration Volume V</i> .

Property Set Representation of Integration Objects

B

Property sets are in-memory representations of integration objects. This appendix describes the relationship between the property set and the integration object. For an overview of property sets, see *Siebel Tools Reference*.

Property Sets and Integration Objects

Many eAI business services operate on integration object instances. Since 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 [Table 24](#).

Table 24. Property Set Node Types

Name	Parent	Value of Type Attribute	Properties	Description
Service Method Arguments	N/A	Ignored	The properties of this property set contain any service specific parameters, such as <i>PrimaryRowId</i> for EAI Siebel Adapter.	This is the top-level property set of a business service’s input or output. The properties of this property set contain any service-specific parameters (for example, <i>PrimaryRowId</i> for EAI Siebel Adapter).
SiebelMessage	Service Method Arguments	SiebelMessage	The properties of this property set contain header attributes associated with the integration object, for example, <i>IntObjectName</i> .	This property set is a wrapper around a set of integration object instances of a specified 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 <i>Hierarchy</i> .
Object List	SiebelMessage	ListOfObjectType	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.

Table 24. Property Set Node Types

Name	Parent	Value of Type Attribute	Properties	Description
Root Component	Object List	<i>Root Component Name</i>	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	<i>ListOfComponent Name</i>	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	<i>Component Name</i>	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 grand-child of the parent component's property set.

Example of a Sample Account

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 a Siebel Tools perspective, looks like that shown in [Figure 46](#).

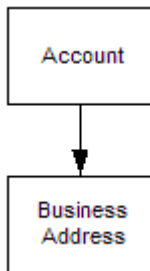


Figure 46. Sample Account Integration Object

[Figure 47 on page 228](#) 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 Representation of Integration Objects

Property Sets and Integration Objects

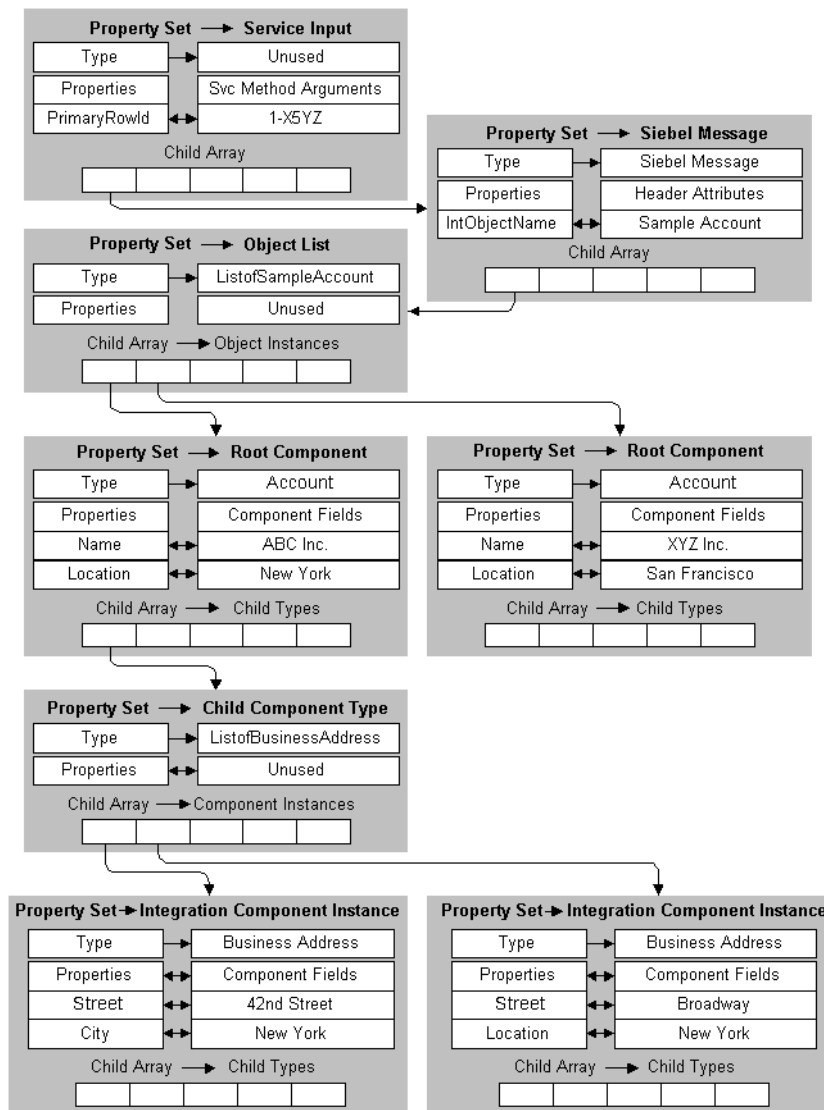


Figure 47. Partial Instance of Sample Account Integration Object

This appendix lists the various inbound and outbound DTDs for the XML Gateway business service.

Outbound DTDs

The following sections contain examples of DTDs representing thtmethodName% request sent from the XML Gateway to the external application.

Delete

```
<!ELEMENT siebel-xmlxt-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

```
<!ELEMENT siebel-xmlxt-fields-req (buscomp, remote-source?)>
<!ELEMENT buscomp (#PCDATA)>
<!ATTLIST buscomp id NMTOKEN #REQUIRED >
<!ELEMENT remote-source (#PCDATA)*>
```

Insert

```
<!ELEMENT siebel-xmlxt-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

```
<!ELEMENT siebel-xmlxt-preinsert-req (buscomp, remote-source?)>
<!ELEMENT buscomp (#PCDATA)>
<!ATTLIST buscomp id NMTOKEN #REQUIRED >
<!ELEMENT remote-source (#PCDATA)*>
```

Query

```
<!ELEMENT siebel-xmlxt-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

```
<!ELEMENT siebel-xmlxt-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

The following sections contain examples of DTDs representing the %methodName% response sent from the external application to the XML Gateway.

Delete Response

```
<!ELEMENT siebel-xmlxt-dekete-ret EMPTY >
```

Init Response

```
<!ELEMENT siebel-xmlxt-fields-ret (support+)>

<!ELEMENT support EMPTY >
```

```
<!ATTLIST support field CDATA #REQUIRED>
```

Insert Response

```
<!ELEMENT siebel-xmlxt-preinsert-ret (row)>
```

```
<!ELEMENT row (value+)>
```

```
<!ELEMENT value (#PCDATA)*>
```

```
<!ATTLIST value field CDATA #REQUIRED >
```

PreInsert Response

```
<!ELEMENT siebel-xmlxt-preinsert-ret (row)>
```

```
<!ELEMENT row (value)*>
```

```
<!ELEMENT value (#PCDATA)*>
```

```
<!ATTLIST value field CDATA #REQUIRED >
```

Query Response

```
<!ELEMENT siebel-xmlxt-query-ret (row*)>
```

```
<!ELEMENT row (value+)>
```

```
<!ELEMENT value (#PCDATA)*>
```

```
<!ATTLIST value field CDATA #REQUIRED >
```

Update Response

```
<!ELEMENT siebel-xmlxt-update-ret (row)>
```

```
<!ELEMENT row (value+)>
```

```
<!ELEMENT value (#PCDATA)>
```

```
<!ATTLIST value field CDATA #REQUIRED >
```


Index

Symbols

%methodName% request, sample inbound
DTDs 231

A

activating fields, about 64
AdminMode user property 69
AllLangIndependentVals user property 70
AllowedIntObjects business service user
property 66
application
external application, about setting
up 183
Siebel Web Service, invoking from 112
arguments
Init method, XML Gateway business
service 173
IsPrimaryMVG 131
AssocFieldName user property
associations with 26
Integration Objects user property 68
Association user property
associations with 26
Integration Objects user properties 67
association, defined 26

B

base object types (table) 18
base table, using Mod Id 141
body data, contents of 17
buscomp Id tag 175
Business Component Id argument 173
Business Component Name argument, XML
Gateway argument 173
business components

association, role of 26
integration restrictions 74
linking 31
multi-value field example 28
multi-value group example 32
relation to business services 77
specialized 166
update permission rules 65
business objects
business service methods, as arguments
to 93
EAI Siebel Adapter, role of 117
external data, creating from 117
integration object maintenance,
about 64
relation to business services 77
structure of 23
user key requirement 38
business service methods
arguments, defining 84
business objects as arguments 93
defining 84
described 79
Business Service Methods screen,
using 87
business service methods, custom
See also virtual business components
about 183
common input parameters (table) 184
connecting methods, list of 184
Delete method, example 186
Error Return property set, example 188
Init method, example 189
Insert method, example 191
output parameters (table) 184

PreInsert method, example 193
 Query method, example 195
 Update method, example 200
 Business Service Simulator, running 89
 business services
 accessing using eScript or Siebel VB 90
 customized business services, type
 of 78
 defined 77
 EAI MIME Hierarchy Converter, creating
 inbound workflow process
 (example) 155
 EAI MIME Hierarchy Converter, creating
 outbound workflow process
 (example) 153
 general uses 77
 importing and exporting 88
 predefined business services, table
 of 219
 property set code example 93
 property sets, about and role of 80
 scripts, defining 85
 Siebel Client, creating in 87
 Siebel Tools, creating process
 overview 82
 Siebel Tools, defining in 83
 Specialized Business Services, about 78
 subsystem, specifying 86
 subsystems (table) 86
 testing 89
 user properties, defining 87
 XML Gateway 169
 BusObjCacheSize argument, about 119,
 121

C

calculated fields 35
 child integration components
 about 24
 structure example 25
 supported operations (table) 129
 child property sets, about 81
 classes
 classes and predefined business
 services 219
 CSSBCVExtern 167
 CSSBCVXMLExten 173
 CSSEAIIDTScriptService 78
 CSSEAISiebelAdapterService 117
 COM connection, external database and
 custom business service 184
 components, defined 15
 concurrency control
 about support of 139
 Account_Organization integration
 component example 144
 configuring 141
 configuring example 143
 Modification IDs, using 140
 Modification Key, about 140
 ContentId property, value and
 description 160
 ContentSubType property 161
 ContentType property 161
 CORBA connection, external database and
 custom business service 184
 CSEEAISiebelAdapterService 66
 CSSBCVExtern class 167
 CSSBCVXMLExten class 173
 CSSBeginEndTransactionService 219
 CSSDataTransformationEngine 221
 CSSDIITransService 220
 CSSEAIIBtsComService 220
 CSSEAIDispatchService 221
 CSSEAIIDTScriptService class 78
 CSSEAIImportExportService 220
 CSSEAIIntObjHierCnvService 221
 CSSEAIMimePropSetService 221
 CSSEAIMimeService 221
 CSSEAINullEnvelopeService 221
 CSSEAIODBCService 220
 CSSEAIQuerySpecService 220
 CSSEAISiebelAdapterService 220
 CSSEAISiebelAdapterService class 117
 CSSEAISMEnvelopeService 221
 CSSEAISqlAdapterService 220

- CSSEAIUtilService 220
- CSSEAIXMLCnvService 221
- CSSEAIXMLPrtService 221, 222
- CSSHTTPTransService 220
- CSSMqAmiTransService 220
- CSSMqSrvTransService 220
- CSSMsmqTransService 220
- CSSSrmService 219
- CSSXMLCnvService
 - XML Converter business service 222
 - XML Hierarchy Converted business service 222
- custom business service
 - Delete method, example 186
 - sample code 203

D

- data and arguments, contrasted 91
- Data Type Definitions
 - See* DTDs
- databases
 - access, controlling 66
 - multi-valued attributes 27
- deactivating fields, about 64
- Delete method
 - custom business service example 186
 - DTD example 229
 - overview 126
 - SearchSpec input method, about and example 134
 - XML code example 129
- Delete Response method, DTD
 - example 231
- DeleteByUserKey argument, about 119
- Display Name field 81
- docking, restrictions on 166
- DoInvokeMethod, about using 117
- DTDs
 - Integration Object Builder wizard, about 22
 - sample inbound DTDs 231

E

- EAI BTS COM Transport business service 220
- EAI Data Mapping Engine business service 221
- EAI Design project, editing integration objects, warning 23
- EAI Dispatch Service business service 221
- EAI DLL Transport business service 220
- EAI HTTP Transport
 - business service, description 220
 - XML Gateway business service, configuring for use by 169
- EAI Import Export business service 220
- EAI Integration Object to XML Hierarchy Converter business service 221
- EAI MIME Doc Converter business service 221
- EAI MIME Hierarchy Converter business service 221
- EAI MQSeries AMI Server Transport, configuring for use by XML Gateway business service 169
- EAI MQSeries AMI Transport business service 220
- EAI MQSeries Server Transport business service 220
- EAI MQSeries Transport, configuring for use by XML Gateway business service 169
- EAI MSMQ Transport business service 220
- EAI MSMQ Transport, configuring for use by XML Gateway business service 169
- EAI Query Spec Service business service 220
- EAI Siebel Adapter
 - concurrency control, about support of 139
 - database access, controlling 66
 - Delete method 126
 - described 117
 - Execute method, overview 126

- Insert method, overview 125
- IsPrimaryMVG argument 131
- language-independent code, using, 138
- method arguments, described (table) 119
- method arguments, locating arguments for (table) 119
- methods, list of 117
- Modification IDs, using 140
- Modification Key, about 140
- multi-value groups 131
- Query method, overview 122
- QueryPage method, overview 123
- run-time events, about using 145
- SearchSpec input method, about and example 134
- Synchronize method, overview 123
- Upsert method, overview 124
- XML example 129
- EAI Siebel Adapter business service 220
- EAI Siebel Wizard
 - about 62
 - integration objects, creating 48
- EAI Transaction Service business service 219
- EAI Utility Service business service 220
- EAI XML Converter business service 221
- EAI XML Read from File business service 222
- EAI XML Write to File adapter, export example 136
- EAI XML Write to File business service 221
- EAISubsys, business service subsystem 86
- Error Return property set example 188
- ErrorOnNonExistingDelete
 - EAI Siebel Adapter Method argument 120
- ErrorOnNonExistingDelete argument, about 119
- error-text tag 180
- eScripts
 - See scripts
- Execute method
 - operations (table) 127
 - overview 126
 - SearchSpec input method, about and example 134
 - specifying and supported parent and child components (table) 128
- export example 136
- Extension property, value and description 160
- extension table, using Mod Id 141
- external application
 - data sharing, process overview 166
 - sample inbound DTDs 231
 - setting up, about 183
- external data source, specifying 168
- External Name user property 26
- external Web Service, invoking using Workflow or Scripting 107

F

- field, defined 15
- FieldDependency
 - Integration Objects user property 69
- fields
 - activating and deactivating 64
 - calculated 35
 - multi-value groups, working with 32
 - picklist, validating and example 33
 - property set fields 80
 - user keys, about 38
- file attachments
 - See also MIME
 - message types 147
 - using, about 147
- force active fields, performance considerations 73
- foreign keys 41
- function code sample 94

G

- guide
 - product modules and options, about 10

revision history 13

H

header data, contents of 17

Hierarchy Parent key, about and
example 44

Hierarchy Root key, about and example 45

history of revisions 13

I

Ignore Bounded Picklist user property 67

Inbound Web Service

creating 97

WSDL file, generating 100

incoming XML format, tags and descriptions
(table) 180

Init method, DTD example 229

Init property set example 189

Init Response method, DTD example 231

Inline XML attachments 148

input parameters, common (table) 184

Input/Output type 85

Insert method, DTD example 230

Insert method, overview 125

Insert property set example 191

instance, defined 16

integration component fields

defined 16

field names, assigning 33

multi-value groups, working with 32

Integration Component Key

See user keys

integration components

activating 63

child components, supported operations
(table) 129

defined 16

deleting during synchronization 60

multi-value groups, working with 32

selecting 49

update permission rules 65

integration messages

body data 17

defined 16

header data 17

Integration Object Builder wizard

about 22

Code Generator wizard 22

EAI Siebel Wizard 63

Generate XML Schema wizard 22

integration components, selecting 49

integration objects, creating 48

user keys, about building 38

user keys, validating 40

integration object instance

actual data, about and diagram 20

defined 16

integration objects

See also child integration components

about 16

base object types (table) 18

best practices and scenarios 74

calculated fields 35

creating 48

defined 16

EAI Design project, editing warning 23

external data, creating from 117

fine tuning practices, list of 51

in-memory updating 57

integration components, deleting during
synchronization 60

maintaining, about 64

many-to-many business component,
creating with 70

metadata, about synchronizing 52

metadata, relation to 19

MIME message objects, creating 149

outbound Web Service, as input

arguments to 104

performance considerations 73

picklist, validating and example 33

primaries, about setting 37

schema, generating 72

SearchSpec field, querying accounts and
addresses based on 135

- simple hierarchy example 226
- structure example 25
- System fields, about treatment of 74
- terminology 15
- testing newly created integration object 71
- update permission rules 65
- updating 53
- user properties, table of 67
- validating 51
- wizards process diagram 21
- integration projects
 - integration objects, use described 22
 - planning 17
- IntObjectName argument
 - described 120
 - locating arguments for 119
- IsPrimaryMVG argument 131

J

- Java class files, generating 22
- joined table, using Mod Id 141

L

- language-independent code
 - list of values, types of 138
 - outbound and inbound direction, about using 138
- LastPage argument, about 121
- LastPage argument, about 119
- links
 - associations, and 26
 - between business components 31
 - update permission rules 65
- LOVs, language-independent code translation 138

M

- many-to-many relationships, virtual business components 166
- MessageId argument
 - described 121

- locating arguments for 119
- metadata
 - defined 15
 - integration objects, updating 53
 - processing example 91
 - relation to integration objects 19
 - synchronizing, integration objects, about 52
- methods
 - business objects as arguments 93
 - business service method arguments, defining 84
 - business services methods, about 79
 - business services methods, defining 84
 - EAI Siebel Adapter method arguments, described (table) 119
 - EAI Siebel Adapter method arguments, locating arguments for (table) 119
 - EAI Siebel Adapter, supported methods 117
 - incoming XML tags by method 180
 - outgoing XML tags by method 175
 - XML Gateway business service method arguments (table) 173
 - XML Gateway business service methods, listed 172
- MIME
 - about 147
 - EAI MIME Doc Converter properties (table) 160
 - inbound workflow process, creating (example) 154
 - integration objects, creating 149
 - messages and hierarchies 159
 - MIME hierarchy, converting to 155
 - outbound workflow process, creating (example) 150
 - workflow process properties, create an outbound workflow process 151
- MIME Doc Converter
 - about 159
 - converting hierarchy to document 153
 - converting to a hierarchy 155

- EAI MIME Doc Converter properties
 - (table) 160
 - properties 161
 - MIME hierarchy
 - converting hierarchy to document 153
 - converting to a hierarchy 155
 - EAI MIME Doc Converter properties
 - (table) 160
 - inbound transformation 158
 - integration object, converting to MIME hierarchy 153
 - MIME Doc Converter 159
 - outbound transformation 156
 - property sets 159
 - MIME Hierarchy Converter
 - business service, creating inbound workflow process (example) 155
 - business service, creating outbound workflow process (example) 153
 - inbound transformation 158
 - outbound transformation 156
 - mobile users and virtual business components 166
 - Modification Key
 - about 140
 - Account_Organization integration component example 144
 - Mod Id field, using for tables 140
 - MVG and MVGAssociation integration components, configuring 141
 - MVG and MVGAssociation integration components, configuring example 143
 - Multi Value Link field 29
 - Multipurpose Internet Mail Extensions. *See* MIME
 - multi-value groups
 - See also* integration objects
 - EAI Siebel Adapter, overview 131
 - example 28
 - field names, assigning 33
 - integration components, creating 32
 - multiple fields 31
 - primary record, setting 132
 - types of 27
 - update permission rules 65
 - virtual business components, restriction 166
 - multi-value links, setting primaries 37
 - multi-valued attributes 27
 - MVG integration components
 - Account_Organization integration component example 144
 - configuring for concurrency control 141
 - example 143
 - MVG integration user property 67
 - MVG. *See* multi-value groups
 - MVGAssociation integration components
 - Account_Organization integration component example 144
 - configuring for concurrency control 141
 - example 143
 - MVGAssociation integration user property 68
 - MVGAssociation user property
 - about 26
 - MVG, creating a Siebel integration component to represent 32
 - MVGFieldName integration user property 68
- N**
- name-value pairs
 - concatenating 170
 - role in property sets 81
 - NewQuery argument 121
 - No envelope business service 221
 - NoDelete user property 68
 - NoInsert user property 68
 - NoUpdate user property 69
 - NumOutputObjects argument
 - described 120
 - locating arguments for 119

O

- ODBC connection, external database and custom business service 184
- Oracle Adapter business service 220
- Oracle Procedure Adapter business service 220
- outbound Web Service
 - integration objects, as input arguments to 104
 - new outbound Web Service, creating manually 102
 - outbound Web Service proxy business service, updating 104
 - run-time data, importing 102
 - WSDL document, reading 100
- outgoing XML format, tags and descriptions (table) 174
- Output Integration Object Name argument, about 120
- output parameters, (table) 184
- Output type 85
- OutputIntObjectName argument, about 119

P

- PageSize
 - EAI Siebel Adapter Method argument 121
 - locating arguments for 119
- parameters
 - common input parameters (table) 184
 - output parameters (table) 184
- Parameters argument, XML Gateway argument 173
- parent business component
 - multi-value group example 32
 - multi-value group field names, assigning 33
- parent integration component
 - about 23
 - child integration component, supported operations (table) 129

- identifying 50
- structure example 25
- performance
 - force-active fields, considerations 73
 - integration object considerations 73
 - picklist considerations 73
- Picklist integration user property 67
- picklists
 - performance considerations 73
 - validating, about and example 33
- PreInsert method, DTD example 230
- PreInsert property set example 193
- PreInsert Response method, DTD example 232
- primaries, about setting 37
- primary business component 23
- primary integration component
 - See parent integration component
- PrimaryRowId argument
 - described 120
 - locating arguments for 119
- process properties
 - importing account information, example 135
- property sets
 - about 223
 - about and role of 80
 - child 81
 - code sample 93
 - Delete method example 186
 - Display Name field 81
 - EAI MIME Doc Converter properties (table) 160
 - Error Return example 188
 - fields 80
 - hierarchy example 226
 - Init example 189
 - Insert example 191
 - integration objects, and 223
 - MIME hierarchy 159
 - nodes types (table) 224
 - PreInsert example 193
 - Query example 195

Update example 200
publishing Inbound Web Services
 creating 97
 WSDL file, generating 100
publishing outbound Web Services
 creating 100
 new outbound Web Service, creating
 manually 102
 outbound Web Service proxy business
 service, updating to point to an
 outbound Web Service 104
 run-time data, importing 102

Q

Query method
 DTD example 230
 overview 122
 SearchSpec input method, about and
 example 134
query operation
 integration component keys, role of 38
 role in integration projects 22
Query property set example 195
Query Response method, DTD
 example 232
QueryByUserKey argument, about 119
QueryPage method
 overview 123
 SearchSpec input method, about and
 example 134

R

Remote Source argument 173
Remote Source user property
 virtual business component 168
 XML Gateway business service 170
REPOSITORY_BC_VIEWMODE_TYPE 66
revision history 13
root component
 See parent integration component
row tag 182
run-time events, about using 145

S

SAPSubsys, business service subsystem 86
schema
 Generate XML wizard 22
 generating 72
scripts
 business service, attaching to 85
 business service, using to access 90
 external Web Service, using to
 invoke 107
SearchSpec argument
 described 121
 locating arguments for 119
SearchSpec input method
 about and example 134
 querying accounts and addresses 135
Search-Spec Node-Type Types, about and
 table 179
Service Name user property
 virtual business component 168
 XML Gateway business service 170
Service Parameters user properties, table
 of 170
Service Parameters user property
 virtual business component 168
 XML Gateway business service 170
Siebel business component, defined 15
Siebel business objects
 defined 15
 structure of 23
Siebel Client, defining business
 services 87
Siebel eScript, using to access a business
 service 90
Siebel integration component
 See integration components
Siebel integration component field,
 defined 16
Siebel integration objects
 See integration objects
Siebel Message envelope business
 service 221

Siebel Message object
 See integration object instance
Siebel Tools
 business services, creating process
 overview 82
 business services, defining 83
 integration objects, creating 48
 user key, identifying 38
 virtual business component,
 creating 167
Siebel VB, using to access a business
 service 90
Siebel Web Service
 See Web Services
SiebelMessage argument
 EAI Siebel Adapter Method
 argument 120
 locating arguments for 119
siebel-xmltext-delete-req tag 175
siebel-xmltext-fields-req tag 175
siebel-xmltext-fields-ret tag 181
siebel-xmltext-Insert-req tag 176
siebel-xmltext-insert-ret tag 181
siebel-xmltext-preinsert-req tag 176
siebel-xmltext-preinsert-ret tag 181
siebel-xmltext-query-req tag 177
siebel-xmltext-query-ret tag 182
siebel-xmltext-status tag 180
siebel-xmltext-Update-req tag 178
siebel-xmltext-Update-ret tag 183
simulation, business service 89
SortSpec argument
 EAI Siebel Adapter Method
 argument 122
 locating arguments for 120
Specialized Business Services, about 78
StartRowNum argument
 EAI Siebel Adapter Method
 argument 121
 locating arguments for 119
Status keys, about 44
status-code tag 180
StatusObject argument

 described 121
 locating arguments for 119
synchronization process
 about 52
 in-memory updating 57
 integration object components,
 deleting 60
 integration objects, updating 53
 role in integration projects 22
 update rules, about 57
Synchronize method, overview 123
System fields, about treatment of 74

T

tables, using Mod Id 141
testing business services 89
transports, used with XML Gateway 169
troubleshooting
 Web Services Tracing, enabling 115

U

Update method
 DTD example 231
Update property set example 200
Update Response method, DTD
 example 232
Upsert method
 overview 124
 XML code example 129
user keys
 building and validating, example 40
 deactivating, warning 43
 defined 38
 definitions, confirming after build 47
 field in Siebel Tools 38
 foreign keys 41
 Hierarchy Parent key, about and
 example 44
 Hierarchy Root key, about and
 example 45
 Integration Component key 38
 locating in Tables screen 40

- Object Builder wizard, about building
 - with 38
- Status keys, about 44
- validity, checking 40
- user properties
 - AssocFieldName 26
 - Association 26
 - business service user properties,
 - defining 87
 - External Name 26
 - integration objects, table of 67
 - MVGAssociation 26
 - virtual business components (table) 168
 - virtual business components, defining
 - for 168

V

- value tag 182
- VBC Compatibility Mode user
 - property 170
- VBCs. *See* virtual business components
- ViewMode argument
 - EAI Siebel Adapter Method
 - argument 122
 - locating arguments for 120
- ViewMode integration object user
 - property 66
- ViewMode user property 70
- virtual business components
 - See also* virtual business components, methods
 - about 163
 - custom code example 203
 - docking restrictions 166
 - external application setup, about 183
 - incoming XML format, tags and
 - descriptions (table) 180
 - mobile users, restriction 166
 - MQSeries, implementing with 171
 - multi-value groups 166
 - new virtual business component,
 - creating 167

- outgoing XML format, tags and
 - descriptions (table) 174
- process overview 166
- Search-Spec Node-Type Types, about and
 - table 179
- specialized business components,
 - restriction 166
- usage and restrictions 165
- user properties (table) 168
- user properties, defining 168
- XML Gateway business service,
 - configuring 170
- virtual business components, methods
 - See also* virtual business components
 - Delete method example 186
 - Error Return property set, example 188
 - Init method, example 189
 - Insert method, example 191
 - PreInsert property set, example 193
 - Query property set, example 195
 - Update property set, example 200
- virtual business services
 - See* business service methods

W

- Web Service Inbound Dispatcher, about
 - using 96
- Web Services
 - external application, invoking from 112
 - Inbound Web Service record,
 - creating 97
 - overview 95
 - scripting, using to invoke 107
 - SOAP messages, about specifying
 - structure in 95
 - standards supported, list of 96
 - troubleshooting 115
 - workflow, using to invoke 107
- Web Services Tracing, enabling 115
- Workflow business service subsystem,
 - described 86
- Workflow Process Manager (Server Request) business service 219

workflows
 external Web Service, using to
 invoke 107
 inbound MIME request 154
 outbound MIME request 150
 policies, about using 145
WSDL file
 external Web service, importing run-time
 data 102
 Inbound Web Services, generating 100
 outbound Web service, based on 100
WSDL Import Wizard
 external Web Services, importing run-
 time data 102
 external WSDL document, using to
 read 100

X

XML
 attribute-named operation,
 specifying 128
 business services, importing 88
 Generate XML Schema wizard 22
 Inline XML attachments 148
 metadata example 92
 upsert and delete code example 129

XML Converter business service 222
XML data, about using Web Service
 Inbound Dispatcher 96
XML format
 incoming tags and descriptions
 (table) 180
 outgoing tags and descriptions
 (table) 174
XML Gateway business service
 See also XML format
 about 169
 configuring 170
 incoming XML tags and
 descriptions 180
 init method arguments 173
 methods (table) 172
 methods arguments (table) 173
 name-value pairs, concatenating 170
 outgoing XML tags and descriptions 175
 sample inbound DTDs 231
 Virtual Business Component,
 implementing with MQSeries 171
XML Hierarchy Converter business
 service 222
XMLCnv business service subsystem 86