Integration Platform Technologies: Siebel Enterprise Application Integration

Siebel Innovation Pack 2013
Version 8.1/8.2
September 2013
Contents

Chapter 1: What’s New in This Release

Chapter 2: Integration Objects
About Integration Object Terminology 13
About Integration Objects 14
About Integration Object Base Object Types 15
About the Difference Between Integration Objects and Integration Object Instances 16
About Integration Object Wizards 16
About the Structure of Integration Objects 18
  Enforcing Cardinality on Existing Integration Objects 20
  About the Cardinality of Child Integration Components 21
  Custom Integration Component Fields 21
  Integration Components and Associations 22
  Multivalue Groups Within Business Components 22
  Setting Primaries Through Multivalue Links 27
  Validation of Integration Component Fields and Picklists 28
  Calculated Fields and Integration Objects 29
  Inner Joins and Integration Components 29
  Defining Field Dependencies 30
  Repository Objects 30
About Integration Component User Properties as Operation Controls 31
About Integration Component Keys 32
  User Keys 32
  Status Keys 36
  Hierarchy Parent Keys 39
  Hierarchy Root Keys 39
About EAI Siebel Adapter Access Control 40

Chapter 3: Creating and Maintaining Integration Objects
About the Integration Object Builder 42
About the EAI Siebel Wizard Business Service 42
Contents

Process of Creating Integration Objects 44
Creating Integration Objects Using the EAI Siebel Wizard Business Service 45
Creating an Integration Object Based on Another Root Business Component 47
Creating an Integration Object with Many-To-Many Relationships 48
Creating Integration Object Instances Programmatically 49
Guidelines for Configuring Integration Objects 52
Validating Integration Objects 52
Testing Integration Objects 53
Deploying Integration Objects to the Run-Time Database 53
About Synchronizing Integration Objects 54
  Synchronization Rules 56
  Updating the Entire Integration Object 56
  Deleting a Component from the Integration Object 58
  Guidelines for Maintaining Integration Objects 60
Synchronizing Integration Objects 61
Resolving Synchronization Conflicts for Integration Objects and User Properties 63
Using Formatted Values in Integration Objects 67
Generating Integration Object Schemas 68
Optimizing the Performance of Integration Objects 69
About Business Component Restrictions for Integration Components 70
Guidelines for Using Integration Components 70

Chapter 4: Business Services
About Business Services 73
  About Creating Business Services 74
  Business Service Structure 74
  Property Sets 75
Creating Business Services in Siebel Tools 76
  Defining a Business Service in Siebel Tools 76
  Defining Business Service Methods 77
  Defining Business Service Method Arguments 77
  Writing Business Service Scripts 78
  Defining Business Service User Properties 78
Creating Business Services in Siebel Business Applications 79
Deploying Business Services as Web Services 80
Exporting and Importing Business Services in Siebel Tools  81
Importing Business Services into Siebel Business Applications  81
Testing Your Business Service in the Simulator  81
About Accessing a Business Service Using Siebel eScript or Siebel VB  82
Business Scenario for the Use of Business Services  83

**Chapter 5: Web Services**

About Web Services  87
About RPC-Literal and DOC-Literal Bindings  89
  RPC-Literal Support  89
  DOC-Literal Support  90
About One-Way Operations and Web Services  90
Invoking Siebel Web Services Using an External System  91
  Publishing Inbound Web Services  91
  Generating a WSDL File  93
  About the Relationship of Port Types and Operations  93
  About Defining the Web Service Inbound Dispatcher  94
  Invoking Web Services on the Siebel Mobile Web Client  95
Consuming External Web Services Using Siebel Web Services  100
  Creating an Outbound Web Service Based on a WSDL File  100
  Creating an Outbound Web Service Manually  102
  Integration Objects as Input Arguments to Outbound Web Services  104
  Web Services Support for Transport Headers  106
  Web Services Support for Transport Parameters  106
Using the Local Business Service  107
  Script Example for a Local Business Service  108
  SOAP Document Generated by the Local Business Service  109
  Using the Local Business Service in an Outbound Web Service  110
About XML Schema Support for the xsd:any Tag  111
  Mapping the xsd:any Tag in the WSDL Import Wizard  111
  Mapping the xsd:any Tag in the XML Schema Wizard  111
Examples of Invoking Web Services  112
About Web Services Security Support  117
  Configuring the Siebel Application to Use the
  WS-Security Specification  117
  About WS-Security UserName Token Profile Support  118
About Siebel Authentication and Session Management SOAP Headers  119
Combinations of Session Types and Authentication Types 124
Enabling Session Management on SWSE 125
Session and Session Token Timeout-Related Parameters 126
Examples of Using SOAP Headers for Authentication and Session Management 127
About Web Services and Web Single Sign-On Authentication 130
About SOAP Fault Schema Support 131
  Handling SOAP Faults in Siebel CRM, Version 8.0 and Earlier 134
  Handling SOAP Faults in Siebel Business Applications, Version 8.1 and Higher 135
About Custom SOAP Filters 136
  Handling Custom Headers Using Filters 136
  Enabling SOAP Header Processing Through Filters 136
  Inputting a SOAP Envelope to a Filter Service 139
About EAI File Streaming 140
  Enabling the EAI Streaming Process 140
  About Inbound EAI Streaming Requests 140
  About Outbound EAI Streaming Responses 141
  About EAI Streaming Criteria 142
About Web Services Cache Refresh 142
Enabling Web Services Tracing 143

Chapter 6: EAI Siebel Adapter Business Service
About the EAI Siebel Adapter Business Service 145
EAI Siebel Adapter Business Service Methods 146
  About the Examples in the EAI Siebel Adapter Business Service Methods Sections 146
  Query Method 147
  QueryPage Method 148
  Synchronize Method 150
  Insert Method 157
  Upsert Method 159
  Update Method 159
  Delete Method 160
  Execute Method 160
EAI Siebel Adapter Business Service Method Arguments 171
About Multivalue Groups in the EAI Siebel Adapter Business Service 175
About Using Effective Dating with Siebel EAI Adapter Business Service 176
Enabling Effective Dating on Fields 177
  Configuring Integration Components for Effective Dating on Fields 177
  How the XML Converter Reads Effective Dating Data from Fields 179
About WSDL Schema Generation for Effective Dating on Fields 180

Enabling Effective Dating on Links 181
   Enabling Effective Dating on Links 182
   About Siebel EAI Adapter Operations for Effective Dating on Links 184

About Using Language-Independent Code with the EAI Siebel Adapter Business Service 185

Siebel EAI and Run-Time Events 186

Guidelines for Using the EAI Siebel Adapter Business Service 187

Troubleshooting the EAI Siebel Adapter Business Service 188

Enabling Logging for the EAI Siebel Adapter Business Service 188

Enabling Siebel Argument Tracing 190

Configuring the EAI Siebel Adapter Business Service for Concurrency Control 191
   Modification Key 191
   Modification IDs 192
   Status IDs 195

Chapter 7: EAI UI Data Adapter Business Service

About the EAI UI Data Adapter Business Service 197

EAI UI Data Adapter Business Service Methods 199
   QueryPage Method 199
   UpdateLeaves Method 204
   InitLeaves Method 206
   InsertLeaves Method 208
   DeleteLeaves Method 212
   Execute Method 213

EAI UI Data Adapter Business Service Method Arguments 215

Exposing Siebel Data to External Applications 216

Chapter 8: Siebel Virtual Business Components

About Virtual Business Components 219
   Using VBCs for Your Business Requirements 220
   Usage and Restrictions for Virtual Business Components 221

Using Virtual Business Components 222
   Creating a New Virtual Business Component 222
   Setting User Properties for the Virtual Business Component 222

XML Gateway Service 224
   XML Gateway Methods 225
Contents

XML Gateway Method Arguments 225
About Handling White Space 226
Examples of the Outgoing XML Format 227
Search-Spec Node-Type Values 230
Examples of the Incoming XML Format 231
External Application Setup 234
Custom Business Service Methods 234
  Common Method Parameters 235
  Business Services Methods and Their Property Sets 235
Custom Business Service Examples 250
  Siebel eScript Business Service Example for a VBC 250
  Siebel VB Business Service Example for a VBC 257

Chapter 9: Siebel EAI and File Attachments
About File Attachments 261
Exchanging Attachments with External Applications 261
Using MIME Messages to Exchange Attachments 262
  Creating an Attachment Integration Object 262
  Creating Workflow Examples 264
About the EAI MIME Hierarchy Converter 269
  Outbound Integration 269
  Inbound Integration 270
About the EAI MIME Doc Converter 270
Using Inline XML to Exchange Attachments 274
  Creating an Attachment 274
  Creating a Test Workflow 275

Chapter 10: External Business Components
Process of Configuring External Business Components 278
  Creating the External Table Definition 279
  Mapping External Columns to Siebel CRM System Fields 284
  Specifying the Data Source Object 285
  Specifying Any Optional Table Properties 285
  Configuring the External Business Component 286
  Specifying Run-Time Parameters 286
Using Specialized Business Component Methods for EBCs 290
Usage and Restrictions for External Business Components 291
About Using External Business Components with the Siebel Web Clients  292
About Overriding Connection Pooling Parameters for the Data Source  293
About Joins to Tables in External Data Sources  293
About Distributed Joins  294
  Configuring Distributed Joins and Federated Fields  295
  Usage and Restrictions for Distributed Joins  295
Troubleshooting External Business Components  296

Appendix A: Predefined EAI Business Services
Predefined EAI Business Services  297

Appendix B: Property Set Representation of Integration Objects
Property Sets and Integration Objects  301
Example Instance of an Account Integration Object  304

Appendix C: DTDs for XML Gateway Business Service
Outbound DTDs for the XML Gateway Business Service  307
Inbound DTDs for the XML Gateway Business Service  309

Index
What’s New in This Release

What’s New in Integration Platform Technologies: Siebel Enterprise Application Integration, Version 8.1/8.2

Table 1 lists the changes described in this version of the documentation to support this release of the software. The new features described in Table 1 are available in Siebel CRM version 8.1.1.11, Siebel CRM version 8.2.2.4, and later.

Table 1. New Product Features in Integration Platform Technologies: Siebel Enterprise Application Integration, Version 8.1/8.2

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Enforcing Cardinality on Existing Integration Objects” on page 20</td>
<td>New topic. It explains how to enforce cardinality on existing integration objects.</td>
</tr>
<tr>
<td>“About WS-Security UserName Token Profile Support” on page 118</td>
<td>Modified topic. It includes corrected namespace information, and information on the requirement to include a specific business service property when enabling the policy.</td>
</tr>
</tbody>
</table>


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

Table 2. New Product Features in Integration Platform Technologies: Siebel Enterprise Application Integration, Version 8.1, Rev D and Version 8.2, Rev A

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Query Method” on page 147</td>
<td>Modified topic. It includes a new note specifying that EAI queries on integration objects do not use a search specification for child integration components when the query obtains the parent integration component.</td>
</tr>
<tr>
<td>“QueryPage Method” on page 148</td>
<td>Modified topic. It includes more information about how the QueryPage method and an example.</td>
</tr>
<tr>
<td>“Enabling Effective Dating on Fields” on page 177</td>
<td>New topic. It explains how to enable effective dating functionality on fields.</td>
</tr>
</tbody>
</table>
What’s New in This Release

Table 2. New Product Features in Integration Platform Technologies: Siebel Enterprise Application Integration, Version 8.1, Rev D and Version 8.2, Rev A

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Enabling Effective Dating on Links” on page 181</td>
<td>New topic. It explains how to enable effective dating functionality on links.</td>
</tr>
<tr>
<td>“Usage and Restrictions for Virtual Business Components” on page 221</td>
<td>Modified topic. In includes a new bullet point discussing limitations when executing queries against virtual business components.</td>
</tr>
</tbody>
</table>
This chapter describes the structure of Siebel integration objects. It describes the Integration Object Builder wizard, which assists you in building your own integration objects based on Siebel objects.

The chapter contains the following topics:

- About Integration Object Terminology on page 13
- About Integration Objects on page 14
- About Integration Object Base Object Types on page 15
- About the Difference Between Integration Objects and Integration Object Instances on page 16
- About Integration Object Wizards on page 16
- About the Structure of Integration Objects on page 18
- About Integration Component User Properties as Operation Controls on page 31
- About Integration Component Keys on page 32
- About EAI Siebel Adapter Access Control on page 40

### About Integration Object Terminology

This chapter describes concepts that are often referred to using inconsistent terminology on different systems. Table 3 has been included to clarify the information in this chapter by providing a standard terminology for these concepts.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>A constituent part of any generic object.</td>
</tr>
<tr>
<td>Field</td>
<td>A generic reference to a data structure that can contain one data element.</td>
</tr>
<tr>
<td>Integration message</td>
<td>A bundle of data consisting of two major parts:</td>
</tr>
<tr>
<td></td>
<td>- Header information that describes what will be done with or to the message itself</td>
</tr>
<tr>
<td></td>
<td>- Instances of integration objects; that is, data in the structure of the integration object</td>
</tr>
<tr>
<td>Integration object</td>
<td>An integration object of any type, including the Siebel integration object, the SAP BAPI integration object, and the SAP IDOC integration objects.</td>
</tr>
</tbody>
</table>
About Integration Objects

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

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

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

When planning your integration project, consider several issues:

- How much data transformation does your message require?
About Integration Object Base Object Types

Each integration object created in Siebel Tools has to be based on one of the base object types presented in 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>
<thead>
<tr>
<th>Base Object Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>For internal use only.</td>
</tr>
<tr>
<td>SAP BAPI Input</td>
<td>Used to represent the input structure of an SAP RFC or BAPI function call.</td>
</tr>
<tr>
<td>SAP BAPI Output</td>
<td>Used to represent the output structure of an SAP RFC or BAPI function call.</td>
</tr>
<tr>
<td>SAP IDOC</td>
<td>Used with the IDOC Adapter and Receiver in version 6.x and 7.0.</td>
</tr>
<tr>
<td>SAP IDOC Adapter</td>
<td>Used to represent an SAP IDOC structure.</td>
</tr>
<tr>
<td>SQL</td>
<td>Used for manually creating integration objects. Only the EAI SQL Adapter</td>
</tr>
<tr>
<td></td>
<td>accepts integration objects of this type.</td>
</tr>
<tr>
<td>SQL Database Wizard</td>
<td>Used by the Database Wizard for the integration object it creates. Only the</td>
</tr>
<tr>
<td></td>
<td>EAI SQL Adapter accepts integration objects of this type.</td>
</tr>
<tr>
<td>SQL Oracle Wizard</td>
<td>Used by the Oracle Wizard for the integration object it creates. Only the</td>
</tr>
<tr>
<td></td>
<td>EAI SQL Adapter accepts integration objects of this type.</td>
</tr>
<tr>
<td>Siebel Business Object</td>
<td>Used by the Integration Object Builder wizard for the integration object it</td>
</tr>
<tr>
<td></td>
<td>creates. The EAI Siebel Adapter accepts only the integration object of this</td>
</tr>
<tr>
<td></td>
<td>type.</td>
</tr>
<tr>
<td>Table</td>
<td>Obsolete.</td>
</tr>
<tr>
<td>XML</td>
<td>Used to represent external XML Schema such as DTD or XSD. For information on</td>
</tr>
<tr>
<td></td>
<td>DTD and XSD, see <em>XML Reference: Siebel Enterprise Application Integration</em>.</td>
</tr>
</tbody>
</table>
About the Difference Between Integration Objects and Integration Object Instances

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

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

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

An integration object instance is actual data organized in the format or structure of the integration object. 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.

About Integration Object Wizards

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

- One that creates integration objects for internal use by the Siebel application
- Others that create integration objects for external systems based on Siebel objects
Figure 2 shows the logic of the Integration Object Builder and Generate XML Schema wizards. The Code Generator wizard (not shown) works in the same manner as the Generate XML Schema wizard, but it generates Java classes.

The following are the integration object wizards:

- **Integration Object Builder wizard.** This wizard lets you create a new object. It supplies the functionality for creating integration objects from Siebel business objects or integration objects, based on the representations of external business objects using XML Schema Definition (XSD) or Document Type Definition (DTD). Access this wizard from the New Object Wizards dialog box in Siebel Tools. After selecting the EAI tab, double-click Integration Object to start the Integration Object Builder wizard.

- **Generate XML Schema wizard.** This wizard lets you choose an integration object and output XML schema in XML Schema Definition (XSD) standard, Document Type Definition (DTD), or Microsoft’s XDR (XML Data Reduced) format. In the Integration Objects list in Siebel Tools, select an integration object. Then click Generate Schema to start the Generate XML Schema wizard.

- **Code Generator wizard.** The third wizard lets you create a set of Java class files based on any available integration object or Siebel business service. In the Integration Objects list in Siebel Tools, select an integration object. Then click Generate Code to start the Code Generator wizard.

**NOTE:** Specific instructions on how to use these wizards appear throughout the Siebel Enterprise Application Integration documentation set where appropriate.
About the Structure of Integration Objects

The Siebel integration object provides a hierarchical structure that represents a complex data type. Most specifically, prebuilt EAI integration objects describe the structure of Siebel business objects, SAP IDOCs, SAP BAPIs, XML, and external data. Most integration projects require the use of an integration object that describes Siebel business objects, either in an outbound direction such as a query operation against a Siebel integration object, or in an inbound direction such as a synchronize operation against a Siebel integration object.

Chapter 3, "Creating and Maintaining Integration Objects" describes how to create integration objects. The initial process of using the Integration Object Builder wizard is essentially the same for every integration object type currently supported.

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

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

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

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

Each child component can have one or more child components. In Siebel Tools, if you look at the integration components for an integration object you have created, then you see that each component can have one or more fields. Figure 3 on page 19 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 3 represents part of the structure of the Account integration object. The Account parent integration component can have both fields and child integration components. Each integration component can also have child integration components and fields. A structure of this sort represents the metadata of an Account integration object. You can choose to inactivate components and fields. By inactivating components and fields, you can define the structure of the integration object instances entering or leaving the system.

The following topics are also discussed:

- “Enforcing Cardinality on Existing Integration Objects” on page 20
- “About the Cardinality of Child Integration Components” on page 21
**Enforcing Cardinality on Existing Integration Objects**

By default cardinality is enforced only on newly created integration objects, but it is enforced for existing integration objects. To enforce cardinality on an existing integration object, you must enable the EnforceCardinality integration object user property.

**To enforce cardinality in a user property**

1. In Siebel Tools, select the integration object on which you want to enforce cardinality.
2. Expand the Integration Object object, then select the Integration Object User Prop object.
3. Add a new user property with the name EnforceCardinality, and set its value to Y.
4. Compile the integration object to your SRF.
About the Cardinality of Child Integration Components

By default each child integration component created in Siebel Tools is assigned a cardinality value of Zero or More. The values “Zero or More”, or “Zero or One” mean that the corresponding integration component is optional. Setting the value to “One” or “One or More” means at least one integration component instance must be included in the hierarchy. When the cardinality value is set to “One” you must have one and only one instance of the integration component in the hierarchy. Table 5 lists possible cardinality values.

Table 5. Cardinality values of Child Integration Components

<table>
<thead>
<tr>
<th>Cardinality</th>
<th>Integration Component Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>An integration component is mandatory and limited to one instance.</td>
</tr>
<tr>
<td>One or more</td>
<td>An integration component is mandatory and must contain at least one instance.</td>
</tr>
<tr>
<td>Zero or more</td>
<td>An integration component is optional, and more than one instance is allowed.</td>
</tr>
<tr>
<td>Zero or one</td>
<td>An integration component is optional, but if there is one present, then the limit is one.</td>
</tr>
</tbody>
</table>

Custom Integration Component Fields

In some cases, you might want to pass custom attributes that are not necessarily part of the actual data but related to the context of the data. You can use various means such as SOAP headers and transport headers to pass such custom fields; however, in Siebel CRM version 8.1 and higher you can have them as part of the integration schema as custom integration component fields.

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

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

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

For example, the Contact business object is partly made up of the Contact and Opportunity business components. The association between these two business components is represented by the Contact/Opportunity link with a value or a table name in the Inter Table column. The Integration Object Builder wizard creates a new integration component for the integration object, based on the Contact business object that represents the association. As shown in Figure 4, the Opportunity integration component has one user property defined: Association, set to a value of Y.

![Integration Components Table](image)

![Integration Component User Props Table](image)

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

Multivalue Groups Within Business Components

Multivalue groups (MVGs) are used within Siebel business components to represent database multivalue attributes. MVGs can be one of two types: regular MVGs or MVG Associations.
An integration object instance most often has multiple integration component instances. For example, an Account can have multiple Business Addresses but only one of these addresses is marked as the primary address. A business requirement might require that only the integration component instance that corresponds to the primary MVG be part of the integration object instance. In relation to Account and Business Addresses this means that only the primary address will be part of the Account integration object instance. The primary address can be obtained by one of the following steps:

- Creating a new MVG on the Account business component that uses a link with a search specification only returning the primary address record.

- Exposing the primary address information on the Account business component level using a join that has the primary ID as source field. Note that in this case the primary address information corresponds to fields on the Account integration component instance and not the fields on a separate Address component instance.

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

**Multivalue Fields in a Business Component**

For example, as illustrated in Figure 5, the Account business component contains a multivalue field, Address Id. The multivalue link property of Address Id has the value Business Address.

![Figure 5. Address Id MVG Field in the Account Business Component](image-url)
Multivalue Links in a Business Component

The Business Address multivalue link associated with the Address Id multivalue field in Figure 5 on page 23 has the value Business Address as its Destination Business Component, as shown in Figure 6.

Fields in a Business Component After Adding a Multivalue Link

The fact that the Business Address multivalue link has Business Address as its Destination Business Component means that there is another business component named Business Address. The Business Address business component contains the fields that are collectively represented by Address Id in the Account business component, as shown in Figure 7.
Graphical Representation of a Business Component and a Multivalue Link

Figure 8 shows a graphical way to represent the relationship between Account business component and the Business Address multivalue link.

Account Business Component (Parent)

<table>
<thead>
<tr>
<th>Address ID</th>
<th>Alias</th>
<th>Full Name</th>
<th>...</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td>123</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Business Address Multivalue Link

Business Address Business Component

<table>
<thead>
<tr>
<th>Account ID</th>
<th>Street Address</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td></td>
<td></td>
</tr>
<tr>
<td>123</td>
<td></td>
<td></td>
</tr>
<tr>
<td>123</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 8. Address Id Field and Business Address Multivalue Link

The more table-like representation in Figure 8 shows how the Business Address multivalue link connects the two business components. The child points to the Business Address business component, which contains the multiple fields that make up the MVG.

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

Creating a Siebel Integration Component to Represent an MVG

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

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

Figure 9 shows an integration component based on an MVG and its user property value in Siebel Tools.
The EAI Siebel Adapter business service must know the names of the MVG fields as they are defined in the parent business component, which in this example is Account, and also the names of the MVG fields as they are known in the business component that represents the MVG, which in this example is Account Business Address. As shown in Figure 10, the integration component fields represent the MVG.

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

### Setting Primaries Through Multivalue Links

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

**NOTE:** It is highly recommended that you use the EAI Siebel Wizard to create integration objects, so that the correct integration components and user properties will also be created. For more information, see “Creating Integration Objects Using the EAI Siebel Wizard Business Service” on page 45.
Validation of Integration Component Fields and Picklists

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

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

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

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

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

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

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

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

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

The value of the PicklistUserKeys user property is a comma separated list of integration component fields that are used to find the matching record in the picklist (for example, 'Account, Account Location' or 'Account Integration Id').
For the EAI Siebel Adapter to use the fields referenced in PicklistUserKeys user property, the fields must be included in the pickmap of the underlying business component field. 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 must contain external names of the fields listed in the PicklistUserKeys user property.

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

**Calculated Fields and Integration Objects**

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

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

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

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

**Inner Joins and Integration Components**

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

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

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

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

Because Project Id is not a required field in the Account business component, not every account in the Siebel Database is associated with a project. So, having Project Name active in the integration component limits the scope of the integration component to only accounts associated with a project. This typically is not desirable, so the wizard inactivates the Project Name field in this example.
If the business requirement is to include the Project Name field, but not to limit the integration component’s scope to only accounts with the project, then you can change the join to S_PROJ in the Account business component to an outer join. For information on joins, see *Using Siebel Tools*.

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

### Defining Field Dependencies

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

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

### Repository Objects

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

Each business component, link, and MVG has properties such as No Update, No Delete, and No Insert. So do integration components, in the form of integration component user properties. These user properties, listed in Table 6, indicate the operations that can and cannot be performed on an object.

Table 6. Operation Controls for Integration Components

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

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

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

There are multiple types of integration component keys:

- **User Key.** See "User Keys" on page 32.
- **Status Key.** See "Status Keys" on page 36.
- **Hierarchy Parent Key.** See "Hierarchy Parent Keys" on page 39.
- **Hierarchy Root Key.** See "Hierarchy Root Keys" on page 39.
- **Modification Key.** See "Configuring the EAI Siebel Adapter Business Service for Concurrency Control" on page 191.

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

User Keys

A user key is a group of fields whose values must uniquely identify a Siebel business component record. During inbound integration, user keys are used to determine whether the incoming data updates an existing record or inserts a new one. The Integration Object Builder wizard automatically creates some user keys based on characteristics discussed in "User Key Generation Algorithm" on page 33. Make sure that the generated user keys match your business requirements; otherwise, inactivate them or add new user keys as appropriate.

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

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

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

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

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

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

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

### User Key Generation Algorithm

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

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

**To find the user keys for a table**

1. Select the Business Component object in the Object Explorer.
   
   The Business Components list appears in the Object List Editor.

2. Select a business component.

3. Click the link in the Table column.
   
   The Tables list appears, displaying the table associated with the business component (for example S_CONTACT).

4. Expand the Tables object in the Object Explorer, and then select User Key.
   
   The User Keys list displays the user keys defined for that table.
For example, as shown in Figure 11, the table S_CONTACT has several user keys.

![User Keys for Table S_CONTACT](image)

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

Each User Key Column child object defined for a given user key must be exposed to the business component in which you are interested. For example, Figure 12 shows three user key columns for the user key S_CONTACT_U1.

![User Key Columns for the S_CONTACT_U1 User Key](image)
If the columns of the user key are exposed in the business component, and those columns are not foreign keys, then the Integration Object Builder wizard creates an integration component key based on the table’s user key. The Integration Object Builder wizard also defines one integration component key field corresponding to each of the table’s user key columns.

The Integration Object Builder wizard builds the integration component keys based on these table user keys. As illustrated in Figure 13, the wizard defines one integration component key for each table user key column.

<table>
<thead>
<tr>
<th>Integration Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>Contact</td>
</tr>
<tr>
<td>Contact</td>
</tr>
<tr>
<td>Contact Business Address</td>
</tr>
<tr>
<td>Contact Position</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Integration Component Keys</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>V70 Wizard-Generated User Key:1</td>
</tr>
<tr>
<td>V70 Wizard-Generated User Key:10</td>
</tr>
<tr>
<td>V70 Wizard-Generated User Key:11</td>
</tr>
<tr>
<td>V70 Wizard-Generated User Key:12</td>
</tr>
<tr>
<td>V70 Wizard-Generated User Key:2</td>
</tr>
<tr>
<td>V70 Wizard-Generated User Key:4</td>
</tr>
<tr>
<td>V70 Wizard-Generated User Key:5</td>
</tr>
<tr>
<td>V70 Wizard-Generated User Key:6</td>
</tr>
<tr>
<td>V70 Wizard-Generated User Key:7</td>
</tr>
<tr>
<td>V70 Wizard-Generated User Key:8</td>
</tr>
<tr>
<td>V70 Wizard-Generated User Key:9</td>
</tr>
</tbody>
</table>

Figure 13. Integration Component Keys for Each Table User Key Column
Each valid integration component key contains fields. For example, as shown in Figure 14, for the Contact integration component, User Key 3 is made up of five fields: CSN, First Name, Last Name, Middle Name, and Personal Contact.

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

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

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

**Status Keys**

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

The status returned is defined in the Integration Component using Status Keys. A Status Key is an Integration Component key of the type Status Key. Fields defined as part of the Status Key are included in the returned StatusObject.
If a Status Key is not defined for the Integration Component then neither the component nor any of its children are included in the returned object:

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

**Status Key Examples**

For example, the AccountAshish integration object has an Account integration component with an integration component key called StatusKeyAshish, with the integration component key fields AshishName, AshishId, and AshishHomePage (Figure 15).

**Example with No StatusObject**

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

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

<PropertySet>

<SiebelMessage MessageId="42-1PGR" IntObjectName="AccountAshish" MessageType="Integration Object" IntObjectFormat="Siebel Hierarchical">

  <ListOfAccountAshish>

    <Account Main_spcPhone_spcNumber="+33123456789" Primary_spcOrganization="Default Organization" Home_spcPage="mycompany.com" Location="France" Name="Ashish 9 Telecom" />

  </ListOfAccountAshish>

</SiebelMessage>

</PropertySet>
```

Figure 15. Status Key
Integration Objects ▪ About Integration Component Keys

Then all of the fields in the integration component are returned:

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

<PropertySet ErrorContextIntComp="" ErrorContextSearchSpec="" OMErrorCode="" PrimaryRowId="42-C739Q" OMErrorSymbol="" ErrorCode="0x0" ErrorSymbol="">

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

Example with StatusObject

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

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

<PropertySet StatusObject="true">

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

Then only the fields in the status key are returned:

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

<PropertySet ErrorContextIntComp="" ErrorContextSearchSpec="" OMErrorCode="" PrimaryRowId="42-C739Q" OMErrorSymbol="" ErrorCode="0x0" ErrorSymbol="">

<SiebelMessage MessageId="42-1PGR" MessageType="Integration Object" IntObjectName="AccountAshish" IntObjectFormat="Siebel Hierarchical">

</SiebelMessage>
</PropertySet>
<ListOfAccountAshish>
  <Account Id="42-C739Q" Home_spcPage="mycompany.com" Name="Ashish 9 Telecom"/>
</ListOfAccountAshish>
</SiebelMessage>
</PropertySet>

For more information on the EAI Siebel Adapter business service, see Chapter 6, "EAI Siebel Adapter Business Service."

Hierarchy Parent Keys

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

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

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

Hierarchy Root Keys

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

**NOTE:** When the business component is hierarchy enabled, then the wizard automatically sets the Hierarchy Parent Key for the complex integration component. To have a business component hierarchy enabled you must set the property Hierarchy Parent Field.
About EAI Siebel Adapter Access Control

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

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

- **ViewMode.** You can specify the visibility mode of business components that the EAI Siebel Adapter uses. This mode is specified as the integration object user property ViewMode. This user property can take different values, as defined by LOV type REPOSITORY_BC_VIEWMODE_TYPE.

**NOTE:** For information on ViewMode, see Siebel Tools Online Help.
3 Creating and Maintaining Integration Objects

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

The chapter contains the following topics:

- About the Integration Object Builder on page 42
- About the EAI Siebel Wizard Business Service on page 42
- Process of Creating Integration Objects on page 44
- Creating Integration Objects Using the EAI Siebel Wizard Business Service on page 45
- Creating an Integration Object Based on Another Root Business Component on page 47
- Creating an Integration Object with Many-To-Many Relationships on page 48
- Creating Integration Object Instances Programmatically on page 49
- Guidelines for Configuring Integration Objects on page 52
- Validating Integration Objects on page 52
- Testing Integration Objects on page 53
- Deploying Integration Objects to the Run-Time Database on page 53
- About Synchronizing Integration Objects on page 54
- Synchronizing Integration Objects on page 61
- Resolving Synchronization Conflicts for Integration Objects and User Properties on page 63
- Using Formatted Values in Integration Objects on page 67
- Generating Integration Object Schemas on page 68
- Optimizing the Performance of Integration Objects on page 69
- About Business Component Restrictions for Integration Components on page 70
- Guidelines for Using Integration Components on page 70
About the Integration Object Builder

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

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

About the EAI Siebel Wizard Business Service

You can use the Integration Object Builder to create integration objects that represent Siebel business objects. During the process of creating a new integration object, described in “Creating Integration Objects Using the EAI Siebel Wizard Business Service” on page 45, you can choose the EAI Siebel Wizard as the business service to help create the object. This wizard understands the structure of Siebel business objects. It returns a list of the available business objects on which you can choose to base your integration object.
The wizard also returns a list of the available components contained within the object you have chosen, shown in Figure 16. When you select certain components in the wizard, you are activating those components in your integration object. Your integration object contains the entire structural definition of the business object you selected in the first wizard dialog box. Only the components you checked, or left selected, are active within your integration object. That means any instances you retrieve of that integration object contains only data represented by the selected components.

![Integration Object Builder - Choose Integration Components](image)

Figure 16. Activated Components in the Contact Integration Object
Creating and Maintaining Integration Objects

Process of Creating Integration Objects

Perform the following tasks to create an integration object:

1. "Creating Integration Objects Using the EAI Siebel Wizard Business Service" on page 45
2. (Optional) Configuring the integration object (see "Guidelines for Configuring Integration Objects" on page 52)
3. "Validating Integration Objects" on page 52
4. Compiling your changes into the SRF file
5. "Testing Integration Objects" on page 53
6. (Optional) "Deploying Integration Objects to the Run-Time Database" on page 53

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

Figure 17. Integration Component Fields in the Contact Integration Object
Creating Integration Objects Using the EAI Siebel Wizard Business Service

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

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

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

This task is a step in “Process of Creating Integration Objects” on page 44.

To create a new Siebel integration object

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

4. On the first page of the Integration Object Builder wizard:
   
   a. Select the project you locked in Step 1.
   
   b. For the source system, select the EAI Siebel Wizard business service.

5. Click Next.
   
   The second page of the Integration Object Builder wizard appears.

   a. Select the source object (business object). This is the object model for the new Siebel integration object. Only business objects with Primary Business Components appear on this picklist.
   
   b. Select the source root (business object component).
Creating and Maintaining Integration Objects ■ Creating Integration Objects Using the EAI Siebel Wizard Business Service

- Type a unique name in the field for the new Siebel integration object and click Next.

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

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

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

**NOTE:** Any component that has a plus sign (+) next to it is a parent in a parent-child relationship with one or more child components. If you deselect the parent component, then the children 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:

- Deselect the Account integration component at the top of the scrolling list. This action deselects the entire tree below Account.
- 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.

7 From the Container Naming Convention drop-down menu, choose either **List Of Prefix** or **Suffixes**.

This allows flexibility when generating XML Schema Definition (XSD) files from integration objects. For example, rather than generating container elements such as xsd:ListOfContact, you can choose to have elements generated named xsd:Contacts.
8 Select the Lower CamelCase for XML Tags check box to use this convention for naming XML tags. CamelCase is a naming convention in which a name is formed of multiple words that are joined together as a single word, with the first letter of each of the multiple words capitalized so that each word that makes up the name can easily be read. The name derives from the hump or humps that seem to appear in any Camel Case name. In Lower CamelCase, the first letter of the name is lowercase, for example myNewIntegrationObject.

The default convention is Upper CamelCase, for example MyNewIntegrationObject.

9 Click Next. The next page displays error or warning messages generated during the process. Review the messages, and take the appropriate actions to address them.

10 (Optional) Select the Deploy the Integration Object check box to deploy the integration object to the run-time database.

For more information, see “Deploying Integration Objects to the Run-Time Database” on page 53.

11 Click Finish.

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

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

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

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

The Integration Component Fields list displays the list of fields for that component. Note the system fields Conflict Id, Created, Id, Mod Id, Updated, operation, and searchspec in the list. This setting prevents the EAI Siebel Adapter Query and QueryPage method from outputting these fields. For more details, see “About Using Language-Independent Code with the EAI Siebel Adapter Business Service” on page 185.

13 When finished, compile the locked project.

### Creating an Integration Object Based on Another Root Business Component

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

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

To create an integration object based on another root business component
1. In Siebel Tools, lock the project containing the business object you want to modify.
2. Modify the business object definition to have that particular root business component as the Primary Business Component.
3. Run the Integration Object Builder wizard and choose the business object you want to use.
4. Undo the changes to the business object definition that you made in Step 2.  
   **NOTE:** This is necessary because unless you are certain about what you are doing in terms of changing the Primary Business Component of the business object, it is recommended that you roll back the changes so that they do not affect any business logic.
5. Compile the locked project.

Creating an Integration Object with Many-To-Many Relationships

To create an integration object with a many-to-many business component
1. In Siebel Tools, create a new project and lock it, or lock an existing project in which you want to create your integration object.
2. From the File menu, choose New Object to display the New Object Wizards dialog box.
3. Select the EAI tab, and then double-click the Integration Object icon.
4. In the Integration Object Builder wizard:
   a. Select the project you locked in Step 1.
   b. Select the EAI Siebel Wizard business service.
5. Click Next and in the second page of the Integration Object Builder wizard:
   a. Select the source object Contact to be the base for the new Siebel integration object.
   b. Type a unique name in the field for the new Siebel integration object, for example Sample Contact M:M, and then click Next.
   c. Select the source root for the new integration object from the list.
6. From the list of components, select Contact and Opportunity.
   **NOTE:** There is also a component named Contact_Opportunity in the list. This component is an MVGAssociation component, and you pick it only if you need this integration object to set the primary opportunity for contact. For information on multivalue groups, see "About Multivalue Groups in the EAI Siebel Adapter Business Service" on page 175.
7 Inactivate all integration component fields in the Contact integration component except First Name, Last Name, Login Name, and Comment. (In this example, these are the only fields you need for Contact.)

8 Inactivate all integration component fields in the Opportunity integration component except Account, Account Location, Budget Amt, Name, and Description. (In this example, these are the only fields you need for Opportunity.)

9 Compile the locked project.

Creating Integration Object Instances Programmatically

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

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

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

- The root property set must have its type set to ListOf concatenated with the integration object name (ListOf\text{IOName}).

- The next property set of the hierarchy must have the root integration component name as its type. The root integration component is the one that has no Parent Integration Component set (\text{RootICName}).

- All other integration components must have the Parent Integration Component set. For those integration components, create a property set with type set to ListOf concatenated with the integration component name (ListOf\text{ICName}) and then add as child to this property set another one with type set to the integration component name.

The following hierarchy demonstrates the rules:

```
ListOfIOName
 RootICName
  ListOfICName1
   ICName1
    ListOfICName1_1
     ICName1_1
  ListOfICName2
   ICName2
```
Figure 18 shows some of the integration components in the hierarchy of the EAI Account integration object.

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

```
List Of EAI Account
    Account
        List Of Account_Business Address
            Account_Business Address
        List Of Contact
            Contact
                List Of Contact_Alt Phone
                    Contact_Alt Phone
```

The following Siebel eScript example creates an instance of the hierarchy shown in Figure 18:

```javascript
// Local variable creation, error handling, and object destruction are omitted for clarity.
psConAltPhone.SetType("Contact_Alt Phone");
psConAltPhone.SetProperty("Alternate Phone ", "555-5555");
```
Creating and Maintaining Integration Objects ■ Creating Integration Object Instances Programmatically

```java
psListOfConAltPhone.SetType("ListOfContact_Alternate Phone");
psListOfConAltPhone.AddChild(psConAltPhone);

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

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

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

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

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

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

psListOfEAIAccount.SetType("ListOfEAI Account");
psListOfEAIAccount.AddChild(psAccount);
...```
Guidelines for Configuring Integration Objects

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

■ In Siebel Tools, inactivate the fields that do not apply to your business requirements.

■ If necessary, activate the fields that have been inactivated by the Siebel Wizard. For information, see Chapter 2, “Integration Objects.”

■ Add the fields that have not been included by the Siebel Wizard, including custom integration component fields. For information on the implications of adding or activating such fields, see the following topics:
  ■ “Custom Integration Component Fields” on page 21
  ■ “Calculated Fields and Integration Objects” on page 29
  ■ “Inner Joins and Integration Components” on page 29

■ Validate the user keys. For information, see Chapter 2, “Integration Objects.”

■ Update the user properties for your integration object to reflect your business requirements. For information, see:
  ■ “Resolving Synchronization Conflicts for Integration Objects and User Properties” on page 63
  ■ “Using Formatted Values in Integration Objects” on page 67

This task is a step in “Process of Creating Integration Objects” on page 44.

Validating Integration Objects

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

This task is a step in “Process of Creating Integration Objects” on page 44.

**To validate your integration object**

1. In Siebel Tools, select your integration object.

2. Right-click the integration object and select Validate.

3. Review the report, and modify your integration object as needed.

**NOTE:** After validation, the integration objects you create in Siebel Tools must be compiled into the SRF file.
Testing Integration Objects

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

This task is a step in “Process of Creating Integration Objects” on page 44.

**To test a newly created integration object**

1. In Siebel Tools, select the Workflow Process object in the Object Explorer.

2. Create a new workflow that runs the EAI Siebel Adapter business service against the new integration object.

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

   ![Workflow Diagram]

   - Start
   - EAI Siebel Adapter
   - EAI XML Write to File
   - End

3. Test the workflow using the Workflow Simulator.

Deploying Integration Objects to the Run-Time Database

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

This saves time by allowing you to modify integration object definitions without having to shut down your production environment, edit integration objects in Siebel Tools, and recompile the SRF.

In version 8.1 and higher, integration objects are read first from the cache and then from the SRF. The deployed integration objects are maintained in the object manager cache so that performance is not slowed by rereading these integration objects from the run-time database.

This task is a step in “Process of Creating Integration Objects” on page 44.

Deploying an Integration Object to the Run-Time Database

The following procedure is used to deploy integration objects that have already been created in Siebel Tools. To deploy an integration object while creating it with the Integration Object Builder wizard, see “Creating Integration Objects Using the EAI Siebel Wizard Business Service” on page 45.

**CAUTION:** If you make changes in Siebel Tools to a deployed integration object, then you must redeploy it. If you do not redeploy it, then the object definitions will differ between Siebel Tools and your production environment, which can cause unexpected application behavior.
To deploy an integration object to the run-time database

1. In the Object Explorer in Siebel Tools, select Integration Object. The Integration Objects list appears.
2. Right-click the integration object to deploy, and then choose Deploy to Runtime Database. The integration object is deployed.
3. In the Siebel client, navigate to the Administration- Web Services screen, Inbound (or Outbound) Web Services view.
4. Click Clear Cache to invalidate the integration object and Web services definitions in the run-time database.

**NOTE:** Object definitions are reloaded when requested in the client.

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

Removing an Integration Object from the Run-Time Database

You can also remove deployed integration objects.

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

1. In the Object Explorer in Siebel Tools, select Integration Object. The Integration Objects list appears.
2. Right-click the integration object to remove, and then choose Undeploy. The integration object is removed from the run-time database.
3. In the Siebel client, navigate to the Administration- Web Services screen, Inbound (or Outbound) Web Services view.
4. Click Clear Cache to invalidate the integration object and Web services definitions in the run-time database.

**NOTE:** Object definitions are reloaded when requested in the client.

About Synchronizing Integration Objects

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

Some examples of these changes are:

- A field removed
A new required field

- A new picklist for a field

- A change of relationship from one-to-many to many-to-many

- An upgrade to a new version of Siebel Business Applications

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

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

The following topics are also covered:

- “Synchronization Rules” on page 56
- “Updating the Entire Integration Object” on page 56
- “Deleting a Component from the Integration Object” on page 58
- “Guidelines for Maintaining Integration Objects” on page 60
Synchronization Rules

During the synchronization process, the wizard follows particular update rules. Consider a simple example involving the Siebel Account integration object with only Contact and its child components marked as active in the object. Figure 19 helps you to visualize this example.

Because the Account component is the parent of Contact, it is also selected, even though you cannot see it in Figure 19.

Updating the Entire Integration Object

Either the business object or the integration object might have changed since the integration object was first created. The Synchronization wizard creates a new object that takes into account any business object and integration object changes.
Figure 20 illustrates how the Synchronization wizard takes into account any changes.
Creating and Maintaining Integration Objects  ■ About Synchronizing Integration Objects

**Figure 21** shows how the resulting integration object is structured after the synchronization.

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

**Deleting a Component from the Integration Object**

If you choose to deselect a component in the Synchronization wizard, then you specify to the wizard to delete the component in the integration object with the matching External Name Context property. The integration object that exists in the database has a component with the same External Name, External Name Sequence, and External Name Context as the unchecked component in the component selection tree.
In Figure 22 on page 59, the Contact_Personal Address in the existing Account integration object is unchecked in the Synchronization wizard tree. This is represented by an X in this figure. Figure 22 illustrates this concept. Figure 23 on page 60 shows the integration object after synchronization.

Figure 22. Deleting a Component from the Integration Object
Creating and Maintaining Integration Objects

About Synchronizing Integration Objects

The component Contact_Personal Address has been deleted. When you use the updated integration object, you cannot pass data for that component between a Siebel application and an external application.

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

Guidelines for Maintaining Integration Objects

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

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

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

Synchronizing Integration Objects

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

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

To update an integration object with updated business object definitions

1. In Siebel Tools, select the integration object you want to update.
2. Click Synchronize in the Integration Objects list.
   
   The Integration Object Synchronize wizard appears.
3. Click on the plus sign to list all the related integration components, as shown in the following figure.

The process of retrieving Siebel integration objects and Siebel business object definitions can take varying amounts of time, depending on the size of the selected objects.
4 Uncheck the boxes beside the objects and components you do not want to include in the synchronization of your Siebel integration object. Note that only the objects that are included in the new integration object are marked.

5 Choose to add new fields as active or inactive and click Next. Inactive is the default.

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

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

   Each added field is checked as to whether or not it is required for use with the integration object.

6 Review the summary. If changes are needed, then click Back and make the needed changes.

7 If no changes are needed, then click Finish to synchronize the Siebel integration object and the Siebel business object.

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

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

   The wizard generally updates the Siebel integration object either by updating the object and its components or by updating some components and deleting others. For information, see “Updating the Entire Integration Object” on page 56 and “Deleting a Component from the Integration Object” on page 58.

8 Copy custom properties and custom user keys as needed. The wizard includes any new fields added to the business object in your integration object for the new version of your Siebel application. All these fields are set to active.

9 Inactivate any new fields that you do not need in a component of your updated integration object.

10 Right-click on your integration object, and select the Validate option to validate your integration object.

   **NOTE:** If you want to synchronize any of the external integration objects, then follow this general procedure to perform a synchronization operation.
Resolving Synchronization Conflicts for Integration Objects and User Properties

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

Table 7 illustrates the behavior of the merging logic for each of the integration object parts that have to be synchronized.

Table 7. Merging Logic Used for Synchronizing Integration Objects

<table>
<thead>
<tr>
<th>Integration Object Metadata</th>
<th>Merging Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objects</td>
<td>Validate that Business Object still exists.</td>
</tr>
<tr>
<td>Components</td>
<td>Present the tree of components based on current business object definition. The components present in the current integration object are checked in the UI tree, other components presented as Inactive. User decides which components to add or delete. This is done by the Synchronization wizard UI.</td>
</tr>
<tr>
<td>Fields</td>
<td>Keep the current integration component fields if still present in the business component, otherwise delete. Add new fields in a way that does not conflict with existing ones (see External Sequence for more information). System fields are created when appropriate (for example, searchspec, IsPrimaryMVG, and operation). If the system field is inconsistent with the integration component definition, then delete it. Active/Inactive. Preserve the current integration component field value unless Business Component Field is Required (field must be present during Insert). Otherwise, new fields are created Inactive.</td>
</tr>
<tr>
<td>XML Properties</td>
<td>Preserve the current integration object values to keep XML compatible. Add new components/fields properties avoiding conflict with existing XML. XML Properties are processed according to the XML sequence. New components/fields that sequence within the parent component element will be higher than current. Reuse existing processing code (and check for correct behavior).</td>
</tr>
<tr>
<td>External Sequence (on components or fields)</td>
<td>Preserve the component or field sequence within the parent component. Set the sequence on new components or fields higher than the existing ones.</td>
</tr>
<tr>
<td>Name</td>
<td>Preserve Names in the current integration object.</td>
</tr>
</tbody>
</table>
Table 7. Merging Logic Used for Synchronizing Integration Objects

<table>
<thead>
<tr>
<th>Integration Object Metadata</th>
<th>Merging Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>User key, Hierarchy key, Other keys (for example, Status Key)</td>
<td>Existing Keys:</td>
</tr>
<tr>
<td></td>
<td>■ Keep existing keys as Active if all the key fields are Active.</td>
</tr>
<tr>
<td></td>
<td>■ Keep existing keys Inactive if Inactive already or make Inactive if any of the fields are Inactive.</td>
</tr>
<tr>
<td></td>
<td>■ If a field is Inactive in an integration component, then make it Inactive in the key. Make the key Inactive.</td>
</tr>
<tr>
<td></td>
<td>■ If a field is not present in an integration component, then delete it from the key. Make the key Inactive.</td>
</tr>
<tr>
<td>New Keys:</td>
<td>■ Create new keys as Inactive.</td>
</tr>
<tr>
<td></td>
<td>■ If any of the key fields are Inactive, then either:</td>
</tr>
<tr>
<td></td>
<td>■ Do not create the key.</td>
</tr>
<tr>
<td></td>
<td>■ Make fields Active in the integration component.</td>
</tr>
</tbody>
</table>

User Properties

Preserve valid cases, remove invalid ones, and generate warnings. See Table 8 on page 64 for more information.

Table 8 shows the logic that is used when synchronizing user properties.

Table 8. Logic Used for Synchronizing User Properties

<table>
<thead>
<tr>
<th>User Property Name</th>
<th>Values (Default is in italics)</th>
<th>Level (Object, Component, or Field)</th>
<th>Merging Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdminMode</td>
<td>Y, N</td>
<td>C, O</td>
<td>Entered by the user; if the value exists, then keep it. Otherwise, the wizard sets the value to N.</td>
</tr>
<tr>
<td>AllLangIndependentVals</td>
<td>Y, N</td>
<td>O</td>
<td>Entered by the user; if the value exists, then keep it. Otherwise, the wizard sets the value to N.</td>
</tr>
</tbody>
</table>
Table 8. Logic Used for Synchronizing User Properties

<table>
<thead>
<tr>
<th>User Property Name</th>
<th>Values (Default is in italics)</th>
<th>Level (Object, Component, or Field)</th>
<th>Merging Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssocFieldName</td>
<td>Any valid field name in the Association business component</td>
<td>F</td>
<td>Siebel Wizard generates the value based on current business component definition. The Wizard overwrites the user change, because in order for the integration component to be functional, the User Property has to be consistent with the business component. (component MVGAssociation is set to Y)</td>
</tr>
<tr>
<td>Association</td>
<td>Y, N</td>
<td>C</td>
<td>Siebel Wizard generates the value based on current business component definition. The Wizard overwrites the user change, because in order for the integration component to be functional, the User Property has to be consistent with the business component.</td>
</tr>
<tr>
<td>EDEnabled</td>
<td>Y, N</td>
<td>F</td>
<td>For each integration component field, the Synchronize wizard adds an integration component field user property named EDEnabled with the value set to Y if the corresponding business component field is effective dating enabled. The wizard will not overwrite user changes.</td>
</tr>
<tr>
<td>FieldDependencyFieldName</td>
<td>Any active integration component name within the same integration component</td>
<td>F</td>
<td>Entered by the user. Keep the current value if valid (if FieldName field is Active).</td>
</tr>
<tr>
<td>ForceUpdate</td>
<td>Y, N</td>
<td>O</td>
<td>Entered by the user. Keep the current value.</td>
</tr>
<tr>
<td>Ignore Bounded Picklist</td>
<td>Y, N</td>
<td>O, C, F</td>
<td>Entered by user, keep if valid (if component Picklist is set to Y).</td>
</tr>
</tbody>
</table>
### Table 8: Logic Used for Synchronizing User Properties

<table>
<thead>
<tr>
<th>User Property Name</th>
<th>Values (Default is in italics)</th>
<th>Level (Object, Component, or Field)</th>
<th>Merging Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>IgnorePermissionErrors</td>
<td>Y, N</td>
<td>C</td>
<td>Entered by the user. Keep the current value.</td>
</tr>
<tr>
<td>OnUpdate, IgnorePermissionErrors</td>
<td></td>
<td></td>
<td>Siebel Wizard generates the value based on the current business component definition. The Wizard overwrites the user change, because in order for integration component to be functional, the User Property has to be consistent with the business component. IsPrimaryMVG system field is created in the merged integration object.</td>
</tr>
<tr>
<td>OnInsert, IgnorePermissionErrors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OnDelete</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MVG</td>
<td>Y, N</td>
<td>C</td>
<td>Siebel Wizard generates the value based on the current business component definition. The Wizard overwrites the user change, because in order for integration component to be functional, the User Property has to be consistent with the business component. IsPrimaryMVG system field is created in merged integration object.</td>
</tr>
<tr>
<td>MVGAssociation</td>
<td>Y, N</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>MVGFieldName</td>
<td>Any valid field name in the MVG business component</td>
<td>F</td>
<td>Siebel Wizard generates the value based on current business component definition. The Wizard overwrites the user change, because in order for integration component to be functional, the User Property has to be consistent with the business component. (component MVG is set to Y)</td>
</tr>
<tr>
<td>NoInsert, NoDelete, NoUpdate, NoQuery, NoSynchronize</td>
<td>Y, N</td>
<td>C, F (NoUpdate)</td>
<td>Entered by the user. Keep the current value.</td>
</tr>
</tbody>
</table>
Using Formatted Values in Integration Objects

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

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

For example, a +55 555 5555 phone number in an English - ENU database is stored as +55555555000 0000, where 000 0000 is the formatting mask.

<table>
<thead>
<tr>
<th>User Property Name</th>
<th>Values (Default is italics)</th>
<th>Level (Object, Component, or Field)</th>
<th>Merging Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Picklist</td>
<td>Y, N</td>
<td>F</td>
<td>Siebel Wizard generated. The user change is kept if valid (if Picklist component). Review the input object for a user property of PICKLIST. Copy from the current field.</td>
</tr>
<tr>
<td>PicklistUserKeys</td>
<td>Any active fields</td>
<td>F</td>
<td>Entered by user, keep only Active fields. User property is valid only if PICKLIST is set to Y on the integration component. If no Active fields left, then remove the user property.</td>
</tr>
<tr>
<td>SuppressQueryOnInsert</td>
<td>Y, N</td>
<td>C</td>
<td>Entered by the user. Keep the current value. When using the Insert method for the EAI Siebel Adapter, if this integration component user property is defined, then the EAI Siebel Adapter will not perform a query before inserting a record.</td>
</tr>
<tr>
<td>ViewMode</td>
<td>All, Manager, Sales Rep, and any others</td>
<td>O</td>
<td>Entered by the user; if the value exists, then keep it. Otherwise, the wizard sets the value to All.</td>
</tr>
</tbody>
</table>
By default, the value of the UseFormattedValues user property is set to N, indicating no formatted values are used. However, you can use scripting or a workflow to configure the EAI Siebel Adapter to force the use of formatted values by setting an integration object’s UseFormattedValues user property to Y. In the previous example, the EAI Siebel Adapter will then return the phone number as +55 555 5555, ignoring the zeroes.

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

## Generating Integration Object Schemas

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

**To generate an integration object schema**

1. In Siebel Tools, select the integration object for which you want to generate a schema.
2. Click Generate Schema to access the Generate XML Schema wizard.
3. Choose the business service to use to generate the schema:
   - **EAI XML DTD Generator.** Generates a Document Type Definition (DTD).
   - **EAI XML XDR Generator.** Generates an XML-Data Reduced (XDR) schema.
   - **EAI XML XSD Generator.** Generates an XML Schema Definition (XSD).
4. Choose an envelope type to use in the schema, either none or Siebel Message Envelope.
5. Choose a location where you want to save the resulting schema file.
6. (Optional) Select the Include length information for String type check box to generate simple types for all string elements in the integration object schema.
7. Click Finish.

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

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

```xml
<!ELEMENT ErrorMessage (#PCDATA)>
<!ELEMENT ErrorCode (#PCDATA)>
```
Optimizing the Performance of Integration Objects

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

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

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

Picklist Validation
Siebel Business Applications have two classes of picklists: static picklists based on lists of values and dynamic picklists based on joins.

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

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

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

**NOTE:** Performing the validation of a bounded picklist in the EAI Siebel Adapter is about 10% faster than performing the validation in the Object Manager.
About Business Component Restrictions for Integration Components

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

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

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

**NOTE:** System fields are read-only. If you attempt to send a message with the value set for a System field, then the setting will be ignored and a warning message will be logged.

Guidelines for Using Integration Components

The following are the guidelines for using integration components:

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

- Design with performance in mind. For more information on performance and using integration objects, see “Optimizing the Performance of Integration Objects” on page 69.

- Design with maintenance in mind. For more information on maintenance, see “Guidelines for Maintaining Integration Objects” on page 60.
Resolve configuration conflicts. During the development of your integration points, you might encounter issues with the configuration of business components that are configured to support interactive GUI usage, but do not satisfy your integration requirements.

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

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

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

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

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

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

**NOTE:** Customized configurations are not automatically upgraded during the Siebel Repository upgrade, so use this option as a last resort.
This chapter outlines the basic concepts of a business service, its structure and purpose, and how you can customize and create your own business service. This chapter also describes how to test your business service before it is implemented. The following topics are included:

- About Business Services on page 73
- Creating Business Services in Siebel Tools on page 76
- Creating Business Services in Siebel Business Applications on page 79
- Deploying Business Services as Web Services on page 80
- Exporting and Importing Business Services in Siebel Tools on page 81
- Importing Business Services into Siebel Business Applications on page 81
- Testing Your Business Service in the Simulator on page 81
- About Accessing a Business Service Using Siebel eScript or Siebel VB on page 82
- Business Scenario for the Use of Business Services on page 83

About Business Services

A business service is an object that encapsulates and simplifies the use of some set of functionality. Business components and business objects are objects that are typically tied to specific data and tables in the Siebel data model. Business services, on the other hand, are not tied to specific objects, but rather operate or act upon objects to achieve a particular goal.

Business services can simplify the task of moving data and converting data formats between the Siebel application and external applications. Business services can also be used outside the context of Siebel EAI to accomplish other types of tasks, such as performing a standard tax calculation, a shipping rate calculation, or other specialized functions.

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

The following topics are also covered:

- "About Creating Business Services" on page 74
- "Business Service Structure" on page 74
- "Property Sets" on page 75
About Creating Business Services

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

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

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

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

- **In Siebel Tools.** Created at design time in Siebel Tools using Siebel VB or Siebel eScript. Design-time business services are stored in the Siebel repository (.SRF), so you have to compile the repository before testing them. When your test is completed, compile and disseminate the SRF to your clients. The business services stored in the repository automatically come over to the new repository during the upgrade process. General business services are based on the class CSSService. However, for the purposes of Siebel EAI, you base your data transformation business services on the CSSEAIDTEScriptService class. For information, see “Creating Business Services in Siebel Tools” on page 76.

- **In Siebel client.** Created at run time in the Siebel client using the Business Service Administration screens. Run-time business services are stored in the Siebel run-time database, so they can be tested right away. The run-time business services have to be migrated manually after an upgrade process. For information, see “Creating Business Services in Siebel Business Applications” on page 79.

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

Business Service Structure

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

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

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

Property sets are used internally to represent Siebel EAI data. A property set is a logical memory structure that is used to pass the data between business services. Figure 24 illustrates the concept of a property set.

The property set contains four parts:

- **Type.** Used to describe what type of object is being represented.
- **Value.** Used to hold serialized data, such as a string of XML data.
  
  **NOTE:** In Siebel Tools, a Value argument to a method is shown with the name of <Value>, including the angle brackets. You can also define a Display Name for the Value argument in Siebel Tools. This Display Name appears in the Siebel Business Process Designer when you are building integration workflows. In this guide, the Display Name Message Text is shown when referring to the Value argument, and the Name <Value> is shown when referring to the Value of the Value argument.

- **Properties.** A table containing name-value pairs. You can use the properties to represent column names and data, field names and data, or other types of name-value pairs.

- **Children.** An array of child-level property sets. You can use the array to represent instances of integration objects. For example, a result set might contain an Account with some set of contact records from the database. Each contact record is represented as a child property set.

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

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

- “Defining a Business Service in Siebel Tools” on page 76.
- “Defining Business Service Methods” on page 77
- “Defining Business Service Method Arguments” on page 77
- “Writing Business Service Scripts” on page 78
- “Defining Business Service User Properties” on page 78

**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 SIEBEL_ROOT\Objects\lang_code 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 Siebel Script Editor within Siebel Tools.

**To define a business service in Siebel Tools**

1. In Siebel Tools, select and lock the project with which you want to associate your business service.
   
   **NOTE:** Each business service must belong to a project, and the project must be locked. For more information, see Using Siebel Tools.

2. Select the Business Services object in the Tools Object Explorer.
   
   The list of predefined business services appears in the right panel.

3. Right-click, and then choose New Record.

4. Type a name in the Name field of the new business service.

5. From the pull-down menu in the Project field, pick the project you locked in Step 1.

6. Choose the appropriate class for your business service from the Class picklist:
Data transformation business services must use the CSSEABSTEScriptService class.
Other business services will typically use the CSSService class.

Step off the current record to save your changes.

## Defining Business Service Methods

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

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

### To define a business service method

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

   The Business Service Methods list appears in the Object List Editor. If you have already defined methods for the selected business service, they appear in the Business Services Methods list.

2. Right-click, and then choose New Record.

3. Type the name of the method in the Name field of the new method.

## Defining Business Service Method Arguments

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

### To define business service method arguments

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

   The Business Service Methods Args list appears in the Object List Editor.

2. Right-click, and then choose New Record.

3. Type the name of the argument in the Name field of the new method argument record.

   **NOTE:** If you plan to use this business service in a Siebel Business Application, then specify the Display Name as well.

4. Enter the data type in the Data Type field.

5. Check the Optional check box if you do not want the argument to be required for the method.
Choose a Type for the argument. Refer to the following table for a list of different types and their descriptions.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>This type of argument serves as input to the method.</td>
</tr>
<tr>
<td>Input/Output</td>
<td>This type of argument serves as both input to the method and output from the method.</td>
</tr>
<tr>
<td>Output</td>
<td>This type of argument serves as output from the method.</td>
</tr>
</tbody>
</table>

Writing Business Service Scripts

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

To write business service scripts

1. In Siebel Tools, select the business service for which you want to write a script.
2. Right-click, and then choose Edit Server Scripts.
   The Siebel Script Editor appears.
3. Select either Siebel eScript or Visual Basic for your scripting language.
4. Select Service_PreInvokedMethod as the event handler.
   NOTE: To write any Siebel VB script in the Business Services, the operating system you are using must support Siebel VB. Siebel VB is not supported in UNIX environments.
5. Type your script into the Script Editor.
   NOTE: Write your business service in Siebel eScript if you want to use the DTE scripts. For information on scripting, see Using Siebel Tools.

Defining Business Service User Properties

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

To define business service user properties

1. With your business service selected in Siebel Tools, expand the Business Service tree in the Object Explorer, and then select Business Service User Prop.
   The Business Service User Props list appears in the Object List Editor.
2 Right-click, then choose New Record.

3 Type the name of the user property in the Name field of the new record.

4 Type a value in the Value field.
   The value can be an integer, a string, or a Boolean.

Creating Business Services in Siebel Business Applications

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

To define a business service in the Siebel application

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

2 Click New to create a new record in the Methods form applet:
   - **Name.** Name of the business service.
   - **Cache.** If checked then the business service instance remains in existence until the user’s session is finished; otherwise, the business service instance will be deleted after it finishes executing.
   - **Inactive.** Check if you do not want to use the business service.

3 Define methods for the business service in the Methods list applet:
   - **Name.** Name of the method.
   - **Inactive.** Check if you do not want to use the method.

4 Define method arguments for the methods in the Method Arguments list applet:
   - **Name.** Name of the method argument.
   - **Type.** The type of the business service method argument. Valid values are Output, Input, and Input/Output.
   - **Optional.** Check if you do not want this argument to be optional.
   - **Inactive.** Check if you do not want to use the argument.

5 From the link bar, select Scripts.

6 Write your Siebel eScript or VB code in the Business Service Scripts list applet.
   **NOTE:** To write any Siebel VB script in the Business Services, the operating system you are using must support Siebel VB. Siebel VB is not supported in UNIX environments.

7 Click Check Syntax to check the syntax of the business service script.
Deploying Business Services as Web Services

You can deploy business services, which you have created in Siebel Tools, as Web services. The Web services can then be consumed by other applications, such as the Siebel Self-Service Applications.

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

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

For more information on Web services, see Chapter 5, “Web Services.” For more information on Siebel Self-Service Applications, see *Siebel Self-Service Application Developer’s Guide*.

**To deploy a business service as a Web service**

1. In the Object Explorer in Siebel Tools, select the Business Service object.
   
   The Business Services list appears.

2. In the Object List Editor, right-click the business service to deploy, and then choose Deploy as Web Service.
   
   The Expose Business Service as Web Service dialog box appears.

3. Specify the following in the dialog box, and then click Finish:
   - **Business service methods to expose.** The operation names for the business service methods are system generated. To edit an operation name, click it in the list.
   - **URL for the Web service.** Replace `<webservice>` with a valid host name and `<lang>` with a valid language code, such as `enu`.
   - **Generate WSDL check box.** To generate a Web Services Description Language (WSDL) file, select the check box, and then choose a location to save the WSDL file.

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

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

**To undeploy a business service**

1. In the Siebel client, navigate to the Administration - Business Services screen.
   
   The Details list appears.

2. Query for the deployed business service, and then select it.

3. Click Delete.

The business service is undeployed.
Exporting and Importing Business Services in Siebel Tools

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

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

Importing Business Services into Siebel Business Applications

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

To import a business service into Siebel Business Applications

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

Testing Your Business Service in the Simulator

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

To run the Business Service Simulator

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

2. In the Simulator list applet, click New to add the business service you want to test.
3. Specify the Service Name and the Method Name.
4. Enter the number of iterations you want to run the business service:
Business Services ■ About Accessing a Business Service Using Siebel eScript or Siebel VB

- Specify the input parameters for the Business Service Method in the Input Property Set applet. Multiple input property sets can be defined and are identified by specifying a Test Case #.
- If the Input Property Set has multiple properties, then these can be specified by clicking on the glyph in the Property Name field. Hierarchical property sets can also be defined by clicking on the glyph in the Child Type field.

5 Click Run to run the business service.

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

The result appears in the Output Property Set applet.

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

About Accessing a Business Service Using Siebel eScript or Siebel VB

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

```javascript
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:

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

### Business Scenario for the Use of Business Services

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

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

```xml
First Name = <FirstName></FirstName>
Last Name = <LastName></LastName>
```

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

```xml
<FirstName>Hector</FirstName>
<LastName>Alacon</LastName>
```

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

### Code Sample Example for Creating a Property Set

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

```javascript
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.

```javascript
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);
}
```

```javascript
Function Service_PreCanBeInvokeMethod(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. The following topics are included:

- About Web Services on page 87
- About RPC-Literal and DOC-Literal Bindings on page 89
- About One-Way Operations and Web Services on page 90
- Invoking Siebel Web Services Using an External System on page 91
- Consuming External Web Services Using Siebel Web Services on page 100
- Using the Local Business Service on page 107
- About XML Schema Support for the xsd:any Tag on page 111
- Examples of Invoking Web Services on page 112
- About Web Services Security Support on page 117
- About Siebel Authentication and Session Management SOAP Headers on page 119
- About Web Services and Web Single Sign-On Authentication on page 130
- About SOAP Fault Schema Support on page 131
- About Custom SOAP Filters on page 136
- About EAI File Streaming on page 140
- About Web Services Cache Refresh on page 142
- Enabling Web Services Tracing on page 143

### About Web Services

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

Business services or workflows in Siebel Business Applications can be exposed as Web services to be consumed by an application, such as one of the Siebel Self-Service Applications. Siebel Web Services framework has an ability to generate WSDL files to describe the Web services hosted by the Siebel application. Also, the Siebel Web Services framework can invoke external Web services. This is accomplished by importing a WSDL document, described as an external Web service, using the WSDL Import Wizard in Siebel Tools.
To specify the structure of XML used in the body of SOAP messages, Web services use an XML Schema Definition (XSD) standard. The XSD standard describes an XML document structure in terms of XML elements and attributes. It also specifies abstract data types, and defines and extends the value domains.

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

**Supported Web Services Standards**
The following Web services standards are supported by Siebel Business Applications:

- Web Services Description Language (WSDL) 1.1
- Web Services Security (WS-Security) based on the clear-text UserName Token mechanism. For information, see the following:
  
  http://schemas.xmlsoap.org/ws/2002/07/secext

  **NOTE:** Passing user credentials in the URL is not supported in Siebel CRM, version 8.2, and version 8.1, Fix Pack 8.1.1.9 and higher.

- Web Services Interoperability (WS-I) Basic Profile 1.0
- Simple Object Access Protocol (SOAP) 1.1
- Hypertext Transfer Protocol (HTTP) 1.1
- Extensible Markup Language (XML) 1.0
- XML Schema
- Extensible Stylesheet Language Transformation (XSLT) 1.0

For more details on supported elements and attributes, see *XML Reference: Siebel Enterprise Application Integration*. For information on supported standards, see:

http://www.w3.org
About RPC-Literal and DOC-Literal Bindings

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

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

RPC-Literal Support

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

The following specifications are required for using RPC-literal:

- **Specification R2717.** An RPC-literal binding in a description must have the namespace attribute specified, the value of which must be an absolute uniformed resource instant (URI), on contained soapbind:body elements.
- **Specification R2729.** A message described with an RPC-literal binding that is a response message must have a wrapper element whose name is the corresponding wsdl:operation name suffixed with the string *Response*.
- **Specification R2735.** A message described with an RPC-literal binding must place the part accessory elements for parameters and return value in no namespace.
- **Specification R2207.** A wsdl:message in a description might contain wsdl:parts that use the elements attribute provided that those wsdl:parts are not referred to by a soapbind:body in an RPC-literal binding.

Making a Web Service an RPC-Literal Web Service

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

*To make a Web service an RPC-literal Web service*

1. Navigate to the Administration - Web Services screen, Inbound Web Services view.
2. Select or add a new namespace from the Inbound Web Services list following the instructions in "Invoking Siebel Web Services Using an External System" on page 91.
3. Create a new inbound service port record in the Service Ports list as indicated in "Invoking Siebel Web Services Using an External System" on page 91.
4. In the Binding column, select SOAP_RPC_LITERAL from the drop-down list.
DOC-Literal Support

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

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

The following is a restriction for using DOC-literal-Specification R2716. A document-literal binding in a description must not have the namespace attribute specified on contained soapbind:body, soapbind:header, soapbind:headerfault, and soapbind:fault elements.

Making a Web service a DOC-literal one is the same as described in ”Making a Web Service an RPC-Literal Web Service” on page 89. When creating the new inbound service port record in the Service Ports list applet, select SOAP_DOC_LITERAL from the drop-down list in the Binding column.

About One-Way Operations and Web Services

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

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

- **Inbound.** If the Business Service Workflow method does not have any output arguments, then it is a one-way operation.

- **Outbound.** If the service proxy method has no output arguments, then it is a one-way operation.

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

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

Defining Support for One-Way Operations

In defining support for one-way operations, the following WS-I Basic Profile specifications are taken into account:

- **Specification R2714.** For a one-way operation, an instance must not return a HTTP response that contains a SOAP envelope. Specifically, the HTTP response entity-body must be empty.

- **Specification R2715.** An instance must not consider transmission of one-way operations complete until a HTTP response status code of either 200 OK or 202 Accepted is received by the HTTP client.

- **Specification R2727.** For one-way operations, an instance must not interpret the HTTP response status code of 200 OK or 202 Accepted to mean the message is valid or that the receiver would process it.
Invoking Siebel Web Services Using an External System

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

**NOTE:** In version 8.0 and higher, you can deploy business services and workflows as Web services and generate WSDL files directly from Siebel Tools. For information on deploying business services, see "Deploying Business Services as Web Services" on page 80. For information on deploying workflows as Web services, see Siebel Business Process Framework: Workflow Guide.

The following inbound Web services topics are covered:

- "Publishing Inbound Web Services" on page 91
- "Generating a WSDL File" on page 93
- "About the Relationship of Port Types and Operations" on page 93
- "About Defining the Web Service Inbound Dispatcher" on page 94
- "Invoking Web Services on the Siebel Mobile Web Client" on page 95

Publishing Inbound Web Services

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

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

**To create an inbound Web service**

1. Navigate to the Administration - Web Services screen, Inbound Web Services view.
2. In the Inbound Web Services list, create a new record:
   - Enter the namespace for your organization’s Web services in the Namespace column.
     **NOTE:** This step is required for generating various XML documents.
   - Enter the name of the inbound Web service in the Name column.
c Select Active in the Status field to enable external applications to invoke the Web service.

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

d (Optional) Enter a description of the Web service in the Comment column.

3 Create an inbound service port record in the Service Ports list:

a Click New and enter the name of the port in the Name column.

b Pick the type of object published. If the required type is not available, then add a new type following Step c through Step f; otherwise, move to Step g.

c Click New and select the implementation type (Business Service or Workflow Process).

d Select the implementation name (the business service or workflow that implements the port type).

e Enter a name for the new type in the Name field and click Save.

f Click Pick in the Inbound Web Services Port Type Pick Applet to complete the process of adding a new Type.

g Select the protocol or transport that will publish the Web service.

h Enter the address appropriate for the transport chosen:

- For the HTTP transport, enter an HTTP address of the Web service to be invoked, such as: 
  \[http://mycompany.com/webservice/orderservice\]

- For the JMS transport, enter the following:
  \[jms://YourQueueName@YourConnectionFactory\]

- For the Local Web Service transport, enter the name of the inbound port.

- For the EAI MSMQ Server transport, enter one of the following:
  \[mq://YourQueueName@YourQueueManagerName\]
  \[msmq://YourQueueName@YourQueueMachineName\]

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

i Select the binding that will publish the Web service.

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

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

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

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

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

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

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

For more information on using the Username/Password Authentication Type, see "About RPC-Literal and DOC-Literal Bindings" on page 89.

---

### Generating a WSDL File

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

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

**To generate a WSDL file**

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

   A WSDL file is generated that describes the Web service.

2. Save the generated file.

3. Import the WSDL to the external system using one of the following utilities:
   - In Microsoft VisualStudio.Net, use the wsd1.exe utility, for example:
     ```
     wsdl.exe /l:CS mywsdlfile.wsdl
     ```
   - In Apache AXIS, use the wsdl2java utility, for example:
     ```
     java org.apache.axis.wsdl.WSDL2Java mywsdlfile.wsdl
     ```
   - In IBM WSADIE, depending on the version, add the WSDL file to the Services perspective and then run the Create Service Proxy wizard.
   - In Oracle JDeveloper, use the Java Web Service from WSDL wizard.

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

---

### About the Relationship of Port Types and Operations

Port types are defined in the Inbound Web Services view, in the Service Ports applet. The Type and Business Service/Business Process Name fields are based on the same dynamic picklist. Opening it displays all the port types. Here you can create or delete port types.
After a port type has been created, you can create the operations that the port type will define. This is done in the Operations applet. Clicking the New button displays any operations that are currently defined for the specified port type. You can expose as many business service methods as you want, but once defined they cannot be deleted or modified through the picklist or through the Operations applet. You can only delete the link between the specified port and operation.

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

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

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

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

**Deleting Operations by Deleting the Port Type**

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

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

**To delete a port type and its operations**

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

**About Defining the Web Service Inbound Dispatcher**

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

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

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

```xml
<?xml version="1.0" encoding="UTF-8" ?>
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:MustUnderstand</faultcode>
      <faultstring>Unable to process SOAP Header child element 'newns:AnotherUselessHeader' with 'mustUnderstand='1''(SBL-EAI-08000)
      </faultstring>
      <detail>
          <siebelf:error>
            <siebelf:errorsymbol/>
            <siebelf:errormsg>Unable to process SOAP Header child element 'newns:AnotherUselessHeader' with 'mustUnderstand='1''(SBL-EAI-08000)
            </siebelf:errormsg>
          </siebelf:error>
        </siebelf:errorstack>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

For more information on SOAP fault handling, see "About SOAP Fault Schema Support" on page 131.

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

NOTE: The functionality described in this topic is provided as ACR 492. For release 8.1 of Siebel Business Applications, implementing this functionality requires installing a patch release and performing postinstallation configuration tasks. For an ACR that is included in a Siebel Fix Pack 8.1.1.x release, see the relevant instructions in the applicable Siebel Maintenance Release Guide on My Oracle Support.

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

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

### Exceptions to Web Service Support

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

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

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

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

### Supported Authentication Formats

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

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

### Application Formats That Are Not Supported

The following authentication formats are not supported:

- Single sign-on (SSO)
- Stateful Web services using separate login and logout requests
- Stateless Web services using a SessionToken
Enabling Web Services on the Siebel Mobile Web Client

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

To enable Web services on the Siebel Mobile Web Client

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EnableWebServices</td>
<td>TRUE</td>
</tr>
<tr>
<td>WebServicesPort</td>
<td>Port number on which to listen. The default is 2330.</td>
</tr>
</tbody>
</table>

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

Starting the siebel.exe Process From the Command Line

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

To start the siebel.exe process from the command line

- Enter the following command:

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

For example:

```
C:\Siebel\80\client\bin\siebel.exe /l enu /c enu\uagent.cfg /u SADMIN /p SADMIN /d Sample /webservice 2330
```

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

Confirming that the siebel.exe Process is Listening

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

To confirm that the siebel.exe process is listening

1. From the DOS prompt, type the following:

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

2. Examine the output for the port number you set in the application configuration file, for example:
TCP mycomputer:2330 mycomputer.mycompany.com:0 LISTENING
LISTENING indicates that the siebel.exe process is listening for Web service calls.

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

Example of Passing User Credentials in the URL
The URL format is:

&Password=<password>

For example:

&Password=<password>

The following is an example of a request:

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

Example of Passing User Credentials in the SOAP Header
The URL format is:

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

For example:

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

The following is an example of a request:

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:asi="http://siebel.com/asi/">
  <soapenv:Header/>
```
Example of Passing User Credentials in the WS-Security Header

The URL format is:

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

For example:

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

The following is an example of a request:

<?xml version="1.0"?>
  <soapenv:Header>
    <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
        <wsse:Username>SADMIN</wsse:Username>
        <wsse:Password Type="wsse:PasswordText">SADMIN</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <asi:SiebelAccountQueryById soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <PrimaryRowId>99-2880A</PrimaryRowId>
    </asi:SiebelAccountQueryById>
  </soapenv:Body>
</soapenv:Envelope>
Consuming External Web Services Using Siebel Web Services

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

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

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

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

The following outbound Web services topics are covered:

- “Creating an Outbound Web Service Based on a WSDL File” on page 100
- “Creating an Outbound Web Service Manually” on page 102
- “Integration Objects as Input Arguments to Outbound Web Services” on page 104
- “Web Services Support for Transport Headers” on page 106
- “Web Services Support for Transport Parameters” on page 106

Creating an Outbound Web Service Based on a WSDL File

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

Note the following restrictions on WSDL import:

- Importing a WSDL containing a cyclic reference, that is, a complex type definition having a member of the same type as itself, is not supported.
- The WSDL Import Wizard expects each schema to have a unique target namespace. Using the same namespace for more than one schema will generate an error.
- Importing a WSDL with a mix of different SOAP operation styles (for example, RPC and Document) within one service port binding is not supported. Modify the WDL to have a service port binding defined for each SOAP operation style.
To create an outbound Web service based on a WSDL file

1. In Siebel Tools, create a new project and lock the project, or lock an existing project.
2. From the File menu, choose New Object to display the New Object Wizards dialog box.
3. Click the EAI tab, and then double-click Web Service.

   The WSDL Import Wizard appears.

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

      **NOTE:** SOAP fault integration objects are prepended with Fault_.

      For more information on SOAP fault integration objects, see “About SOAP Fault Schema Support” on page 131.

4. Click Next.

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

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

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

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

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

This procedure generates three objects in the Siebel repository:

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

  **NOTE:** For RPC services, the order of input arguments is important. You can set the order through the Preferred Sequence property of the business service method argument in Siebel Tools. By specifying this parameter, the outbound dispatcher makes sure that the sequence parameters for an operation are in the correct order. The Preferred Sequence property is only supported with outbound services.
Integration objects, representing input and output parameters of the service methods, if any of the operations require a complex argument (XML Schema) to be passed. If the service does not use complex arguments, then no integration object definitions will be created.

A Web service administration document (XML file) containing the run-time Web service administration data to be imported into the Siebel Web Client, using the Outbound Web Services view of the Administration - Web Services screen.

The purpose of the document is to allow administrators to modify run-time parameters such as the URL and encoding rules. The data contained within the document is used by the Web Services Dispatcher to assemble the SOAP document, to set any HTTP headers required (for example, soapAction), and to route the request to the correct URL. For information, see “To import run-time data about external Web services” on page 102.

Importing Run-Time Data About External Web Services

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

To import run-time data about external Web services

1. Restart the Siebel Server (or Siebel 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 have to update your SRF file at design time. However, the service definition must exist in the SRF file during run time.

2. Navigate to the Administration - Web Services screen, Outbound Web Services view.

3. In the Outbound Web Services list applet, click Import to display 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.

Creating an Outbound Web Service Manually

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

To manually create a new outbound Web service

1. Navigate to the Administration - Web Services screen, Outbound Web Services view.

2. In the Outbound Web Services list applet, create a new record:
   a. Enter the namespace of the Web service in the Namespace column.
   b. Enter the name of the Web service in the Name column.
Web Services

Consuming External Web Services Using Siebel Web Services

Select Active or Inactive in the Status field.

Enter a description of the Web service in the Comment column.

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

In the Service Ports list applet, create a new outbound service ports record:

a Enter the name of the Web service port in the Name column.

b Select a transport name for the protocol or queuing system for the Transport.

Enter the address for the transport chosen to publish the Web service:

- The URL format to publish using HTTP is:
  

  where:

  - webserver is the name of computer where the Siebel Web Server is installed
  - lang is the default language of the Object Manager that handles the request

- The format to publish using the EAI JMS Transport is:
  
  jms://queue name@connection factory

  where:

  - queue name is the Java Naming and Directory Interface (JNDI) name of the queue
  - connection factory is the JNDI name of the JMS connection factory

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

- The format to publish over the EAI MQSeries or EAI MSMQ transport is:
  
  mq://queue name@queue manager name
  
  msmq://queue name@queue machine name

  where:

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

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

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

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

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

- For the Local Web Service transport, enter the name of the inbound port.
5 Select the binding that will publish the Web service.

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

Use the Property Set binding when the input property set to the proxy service is forwarded without changes to the destination address. This is intended primarily for use in combination with the Local Workflow Process or Local Business Service transport to avoid the overhead of processing XML.

6 Enter a description of the port in the Comment column.

7 In the Operations list applet, create a new operation record for the new service port you created in Step 3 on page 103:

   a Select the name of the business service method in the Method Display Name column to complete the process.

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

      **NOTE:** For more information on using the Username/Password Authentication Type, see “About Web Services Security Support” on page 117.

8 Generate the WSDL file. For information, see “Generating a WSDL File” on page 93.

**Updating the Outbound Proxy Business Service**

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

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

1 In Siebel Tools, select the outbound Web service proxy business service you want to use to call your outbound Web service.

2 Add the following user properties for this business service and set their values based on the outbound service port of your Web service:

   - siebel_port_name
   - siebel_web_service_name
   - siebel_web_service_namespace

**Integration Objects as Input Arguments to Outbound Web Services**

It is recommended that the property set used as an input argument to the outbound Web service have the same name as the input argument's outbound Web service proxy.
You can do this using one of the following options:

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

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

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

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

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

    for (var i = 0; i < Inputs.GetChildCount(); i++) {
        if (Inputs.GetChild(i).GetType() == "SiebelMessage") {
            childPS = Inputs.GetChild(i);
        }
    }

    childPS.SetType("myBusSvcMethod:myIntegrationObject");
    newInputPS = TheApplication().NewPropertySet();
    newInputPS.AddChild(childPS);

    svc = TheApplication().GetService("myBusSvc");
    svc.InvokeMethod("myBusSvcMethod", newInputPS, Outputs);
    return (CancelOperation); // must use CancelOperation with custom methods
}
```
Web Services Support for Transport Headers

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

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

- Name: siebel_transport_header:headerName
- Value: Header value

Table 9 shows examples of input arguments for transport headers.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>siebel_transport_header:UserDefinedHeader</td>
<td>myData</td>
</tr>
<tr>
<td>siebel_transport_header:Authorization</td>
<td>0135DFDJKLJ</td>
</tr>
</tbody>
</table>

Web Services Support for Transport Parameters

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

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

- Name: siebel_transport_param:parameterName
- Value: Parameter value

Table 10 shows examples of input arguments for transport parameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>siebel_transport_param:HTTPRequestMethod</td>
<td>HTTP method to use with the data request, such as Post or Get</td>
</tr>
<tr>
<td>siebel_transport_param:HTTPRequestURLTemplate</td>
<td>Template for the request URL, which is the address to which the data is sent or from which a response is requested, for example: <a href="http://mycompany.com/">http://mycompany.com/</a>*</td>
</tr>
</tbody>
</table>

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

In many instances, Web services use specialized SOAP headers for common tasks such as authentication, authorization, and logging. To support this common Web service extensibility mechanism, the Local Business Service transport for outbound Web services can be used. When specified as a transport, the Web services infrastructure will route the message to the specified business service for additional processing and delivery to the Web service endpoint as shown in the top half of Figure 25.

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

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

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

The following local business service topics are also discussed:

- “Script Example for a Local Business Service” on page 108
- “SOAP Document Generated by the Local Business Service” on page 109
- “Using the Local Business Service in an Outbound Web Service” on page 110
Script Example for a Local Business Service

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

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

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

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

// SOAP header property set. Once this is complete, add the SOAP header as a child of the Input property set (which contains the SOAP:body).
soapHdr.InsertChildAt(profileHeader, 0);
Inputs.InsertChildAt(soapHdr, 0);

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

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

// Invoke the Web service using the standard HTTP protocol.
httpSvc.InvokeMethod("SendReceive", outPs, hpOut);

// Convert the SOAP document to a property set using the XML Converter, returning
// the SOAP header and SOAP body.
xmlCtr.InvokeMethod("XMLToPropSet", hpOut, Outputs);

After you have created your business service, compile it into the Siebel SRF.

**SOAP Document Generated by the Local Business Service**

The following example displays the resulting SOAP document generated by the "Script Example for a Local Business Service" on page 108. The addition of the authHeader element to the SOAP header corresponds to the structure defined in the sample code sections that populate the SOAP header and profileHeader elements.

```xml
<?xml version="1.0" encoding="UTF-8"?>
```
Using the Local Business Service in an Outbound Web Service

You use the Outbound Web Services view in the Administration - Web Services screen to configure an outbound Web service to use the local business service created by "Script Example for a Local Business Service" on page 108.

To use the local business service in an outbound Web service

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport</td>
<td>Local Business Service</td>
</tr>
<tr>
<td>Address</td>
<td>Name of the local business service</td>
</tr>
</tbody>
</table>
Restart the Siebel Server component to allow the changes to take effect.

About XML Schema Support for the xsd:any Tag

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

Mapping the xsd:any Tag in the WSDL Import Wizard

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

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

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

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

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

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

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

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

Mapping the xsd:any Tag in the XML Schema Wizard

For the case of the XML Schema Wizard, there is only one possible mapping for the xsd:any tag, namely as an integration component.
The xsd:any tag can contain a namespace attribute. If the value for that attribute is known, then one or more integration components or even an integration object can be created. If the value for that attribute is not known, then an error will be returned to the user saying that the integration object cannot be created for a weakly typed schema.

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

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

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

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

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

**Examples of Invoking Web Services**

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

**Invoking an External Web Service Using Workflow or Scripting**

As illustrated on Figure 26 on page 113, the following steps are executed to call an external Web service:

1. The developer obtains the Web service description as a WSDL file.
2. The WSDL Import Wizard is called.
3. The WSDL Import Wizard generates definitions for outbound proxy, integration objects for complex parts, and administration entries.
4. The Outbound Web Service proxy is called with the request property set.
5. The request is converted to an outbound SOAP request and sent to the external application.
6. The external application returns a SOAP response.
The SOAP response is converted to a property set that can be processed by the caller, for example, Calling Function.

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

```javascript
SiebelMessagePS.SetProperty("MessageType", "Integration Object");
SiebelMessagePS.SetProperty("IntObjectName", "Sample Household Interface");
SiebelMessagePS.SetProperty("IntObjectFormat", "Siebel Hierarchical");
SiebelMessagePS.AddChild(psListOfSampleHouseholdInterface); // psListOfSampleHouseholdInterface
psInput.AddChild(SiebelMessagePS);

function Service_PreCanInvokeMethod (MethodName, &CanInvoke) {
    if (MethodName == "invoke") {
        CanInvoke = "TRUE";
        return (CancelOperation);
    }
}
```

Figure 26. Invoking an External Web Service
else
    return (ContinueOperation);
}
}
}

else
    return (ContinueOperation);
}
}

return (CancelOperation);
}
catch (e) {
    TheApplication().RaiseErrorText(e);
    return (CancelOperation);
}

return (CancelOperation);
}

</ListOfSampleHouseholdInterface>
</SiebelMessage>
Invoking a Siebel Web Service from an External Application
As illustrated in Figure 27 on page 115, the following steps are executed to invoke a Siebel Web service from an external application:

1. The WSDL document for an active Web service is published in the Siebel Inbound Web Services view. To allow processing of the Web service requests, the developer has to make sure:
   a. The Web Server and the Siebel Server are up and running.
   b. The appropriate setup is done in the Siebel Server.

2. In the external application, the WSDL document is imported to create a proxy that can be used to invoke the Siebel Web service from Step 1.

3. The external application sends the SOAP request into the Siebel application.

4. The Web Service Inbound Dispatcher converts the SOAP request to a property set. Depending on the inbound Web service configuration, the property set is passed to a business service or a business process.

5. The property set is returned from the business service or business process to the Web Service Inbound Dispatcher.

6. Response is converted to a SOAP message and sent back to the invoking external application.

Figure 27. Invoking a Siebel Web Service
The following is an example of invoking a Siebel-published Web service using .NET.

```csharp
// Removed using declaration

namespace sieOppClnt {
        public sieOppClnt() {
            InitializeComponent();
        }

        // WEB SERVICE CLIENT EXAMPLE

        /* The optyQBE returns a list of opty based upon the required input params. Because
         * the input to the Siebel opty.QueryByExample method uses an Input/Output param,
         * ListOfInterOptyInterfaceTopElmt will be passed by ref to Siebel. To add the Siebel
         * Opportunity Web Service definition to the project, the wsd1.exe utility was run
         * to generate the necessary helper C# class for the service definition. */

        [WebMethod]
        public ListOfInterOptyInterfaceTopElmt optyQBE(string acctName, string acctLoc,
            string salesStage) {
            Siebelopty svc = new Siebelopty();
            ListOfInterOptyInterfaceTopElmt siebelMessage = new ListOfInterOptyInterfaceTopElmt();
            ListOfInterOptyInterface optyList = new ListOfInterOptyInterface();
            opty[0].opty = new opty[1];
            opty[0] = new opty[1];
            opty[0].Account = acctName;
            opty[0].AccountLocation = acctLoc;
            opty[0].SalesStage = salesStage;

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

            optyList.opty = opty;
            siebelMessage.ListOfInterOptyInterface = optyList;

            // Invoke the QBE method of the Siebel Opportunity business service

            svc.SiebeloptyQBE(ref siebelMessage);

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

            return siebelMessage;
        }
    }
}
```
About Web Services Security Support

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

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

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

Configuring the Siebel Application to Use the WS-Security Specification

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

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

1. Set the UseAnonPool parameter in the eapps.cfg (SWE plug-in) file under [/eai_anon_enu] as follows:
   ```
   UseAnonPool = TRUE
   ```
2. Start the Siebel Server.
3. Navigate to the Administration - Server Configuration screen, Enterprises view, and then Profile Configuration.
4. In the Profile Configuration list, query the Alias field for SecureWebService.
5. Make sure that the SecureWebService profile (named subsystem) has parameters with the following values:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Alias</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Method to Execute</td>
<td>DispatchMethod</td>
<td>Dispatch</td>
</tr>
<tr>
<td>Service to Execute</td>
<td>DispatchService</td>
<td>Web Service Inbound Dispatcher</td>
</tr>
<tr>
<td>Impersonate</td>
<td>Impersonate</td>
<td>True</td>
</tr>
</tbody>
</table>

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

   ```
   http://myserver/eai_anon_enu/start.swe?SWEExtCmd=Execute
   &SWEExtSource=SecureWebService
   ```
About WS-Security UserName Token Profile Support

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

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

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

```xml
    <wss:Username>Username</wss:Username>
    <wss:Password Type="wss:PasswordText">Password</wss:Password>
  </wss:UsernameToken>
</wss:Security>
```

To enable the `oracle/wss_username_token_service_policy`, the corresponding business service for the imported outbound WSDL must include a specific business service property, which is as follows:

- **Property Name:** auth_policy
- **Value:** `oracle/wss_username_token_service_policy`. If the property is not specified, then the default value is `oracle/no_authentication_service_policy`.

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

About Support for the UserName Token Mechanism

Support for the UserName Token mechanism includes the following:

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

**NOTE:** Passing user credentials in the URL is not supported in Siebel CRM, version 8.2, and version 8.1, Fix Pack 8.1.1.9 and higher.
Using the UserName Token for Inbound Web Services
The Inbound Web Services view provides an interface for associating operations with authentication types. The names of the operations must be globally unique. The applet shown in Figure 28 can be defined as requiring a UserName Token with username and password provided in clear text.

![Figure 28. Inbound Web Services View and the UserName Token](image)

**NOTE:** If you want to use Siebel Authentication and Session Management SOAP headers, then set the authentication type to None. For more information, see “About Siebel Authentication and Session Management SOAP Headers” on page 119.

Using the UserName Token for Outbound Web Services
Each Web service operation in the Outbound Web Services list applet might be tied to an authentication type by selecting from the Authentication Type picklist from the Outbound Web services list, as shown in Figure 29.

![Figure 29. Outbound Web Services Operations PickList](image)

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

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

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

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

You can use the SOAP headers listed in Table 11 on page 121 to call different types of sessions and pass authentication credentials:

**NOTE:** The values entered are case insensitive.
Table 11. Siebel Session Management and Authentication SOAP Headers

<table>
<thead>
<tr>
<th>SOAP Header Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessionType</td>
<td>You use the SessionType SOAP header to define the type of session. Valid values are None, Stateless, Stateful, and ServerDetermine:</td>
</tr>
<tr>
<td></td>
<td>- <strong>None.</strong> A new session is opened for each request and then closed after a response is sent out. The SessionType none might or might not include UsernameToken and PasswordText SOAP headers. When UsernameToken and PasswordText SOAP headers are included, these credentials are used for authentication.</td>
</tr>
<tr>
<td></td>
<td>If the UsernameToken and PasswordText SOAP headers are excluded from the SOAP header, then anonymous login is assumed. The anonymous login requires additional configuration in the Siebel Web Engine (eapps.cfg) and Named Subsystem configuration (AllowAnonymous equals (=) True, Impersonate equals (=) False).</td>
</tr>
<tr>
<td></td>
<td>For more information about configuring Anonymous login, see <em>Siebel Security Guide</em>.</td>
</tr>
<tr>
<td></td>
<td>- <strong>Stateless.</strong> A new session is opened for an initial request and the session remains open for subsequent requests. Relogin occurs automatically (transparent to the user) if the session is closed. UsernameToken and PasswordText must be included as SOAP headers in the initial request to open a stateless session.</td>
</tr>
<tr>
<td></td>
<td>Stateless session management is the best method to use for high-load Web service applications. Using Stateless mode, the application provides the username and password only once, that is for the initial request. A session is opened on the server and is dedicated for this user.</td>
</tr>
<tr>
<td></td>
<td>In the response Siebel Business Applications return the SessionToken, which is an encrypted string containing the information about username, password, and timestamp. For subsequent requests the application must use the SessionToken to reuse the session.</td>
</tr>
<tr>
<td></td>
<td>For security reasons SessionTokens are regenerated for each response. The application must provide the last received SessionToken for the next request.</td>
</tr>
<tr>
<td></td>
<td>The SessionToken-Siebel session map is maintained in the Siebel Web Server Extension (SWSE); based on the SessionToken value SWSE sends the request to the correct Siebel session (task).</td>
</tr>
<tr>
<td></td>
<td>Although the session is persistent, authentication happens for each request (SWSE decrypts the UserName and Password from the SessionToken).</td>
</tr>
</tbody>
</table>
About Siebel Authentication and Session Management SOAP Headers

Table 11. Siebel Session Management and Authentication SOAP Headers

<table>
<thead>
<tr>
<th>SOAP Header Block</th>
<th>Description</th>
</tr>
</thead>
</table>
| SessionType       | ■ **Stateful.** A new, dedicated session is opened for an initial request and the session remains open for subsequent requests. Relogin does not occur automatically if the session is closed. UsernameToken and PasswordText must be included as SOAP headers in the initial request to open a stateful session.  
As with Stateless sessions, Siebel Business Applications return the SessionToken in the response. For subsequent requests the application must use the SessionToken to reuse the session.  
Unlike Stateless sessions, transparent failover (automatic relogin) is not supported. This is because Stateful sessions might have state information stored that makes it mandatory to connect to the same task for each request.  
■ **ServerDetermine.** A new session is established to Siebel, and a series of subsequent requests is served. The Siebel Server is free to multiplex the session to serve other users if possible, but the client is free to make stateful calls to Siebel. Failover is not supported for this mode.  
ServerDetermine provides the most flexibility: the session can be dedicated or not. If the number of users increases and resources must be recovered, then the session state is written to the database so that it can be restored. The session can then serve other users.  
If SessionType is absent, then the default value is None, and the session will be closed after the request is processed. |
| UsernameToken     | You use the UsernameToken SOAP header to send the Login ID to the Siebel Server. |
About Siebel Authentication and Session Management SOAP Headers

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

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

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

<table>
<thead>
<tr>
<th>SOAP Header Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PasswordText</td>
<td>You use the PasswordText SOAP header to send the password used by the login ID to the Siebel server. If using Web single sign-on (SSO), then use the Siebel trust token value in PasswordText instead of the password.</td>
</tr>
<tr>
<td>SessionToken</td>
<td>Session tokens are used with stateless requests. They are sent and received using the SessionToken SOAP header. After receiving an initial request with valid authentication credentials and a session type set to Stateless, the Siebel Server generates a session token and includes it in the SOAP header of the outbound response. The session token is encrypted and contains a session ID and user credentials. The custom Web application uses the session token for subsequent requests. The Session Manager on the SWSE extracts a session ID and user credentials from the session token, and then passes the information to the Siebel Server. The session ID is used to reconnect to an existing session or automatically log in again if the session has been terminated. <strong>NOTE:</strong> Reconnecting or automatic logging in again will only happen if the token has not timed out. If it times out, then the user must manually log in again. Token timeout must be greater than or equal to session timeout. For more information on session token timeout, see “Session and Session Token Timeout-Related Parameters” on page 126. However, the session token must be changed to the new one sent on every response. The session token has a maximum time to live, which can invalidate it even if its timeout (for being inactive) has not been reached. Always get the newest session token returned by the response and use it on the next request. The same session token must not be used by concurrent requests, because having multiple requests point to the same session can cause errors.</td>
</tr>
</tbody>
</table>

For examples of using SOAP headers for session management and authentication, see "Examples of Using SOAP Headers for Authentication and Session Management" on page 127.
Combinations of Session Types and Authentication Types

Table 12 summarizes the combinations of authentication types and session types.

Table 12. Summary of Authentication Types and Session Types

<table>
<thead>
<tr>
<th>Authentication Type</th>
<th>Session Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>None</td>
<td>A single request is sent with an anonymous user login, and the session is closed after the response is sent out. In order for the anonymous session to be identified by the SWSE Plug-in, UsernameToken and PasswordText must be excluded in the SOAP headers.</td>
</tr>
<tr>
<td>Username and password</td>
<td>None</td>
<td>A single request is sent with the username and password used to log in, and the session is closed after the response is sent out.</td>
</tr>
<tr>
<td>Username and password</td>
<td>Stateless</td>
<td>The initial request to log in establishes a session that is to remain open and available for subsequent requests. Username/password are used to log in and a session token is returned in a SOAP header included in the outbound response. The session remains open.</td>
</tr>
<tr>
<td>Session token (stateless)</td>
<td>Stateless</td>
<td>Request to reconnect to an established session, using the information contained in the session token. If the session has been closed, then automatic relogin occurs. The Siebel servers include the session token in the SOAP header of the response. The session remains open.</td>
</tr>
<tr>
<td>Session token (stateless)</td>
<td>None</td>
<td>When a SOAP header carries a session token and has the session type set to None, then the Session Manager on the SWSE closes (logs out) of this session, and invalidates the session token. The session token is not used after the session is invalidated.</td>
</tr>
</tbody>
</table>

For examples that illustrate some of these combinations, see "Examples of Using SOAP Headers for Authentication and Session Management" on page 127.
Enabling Session Management on SWSE

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

```
http://mywebserver/EAIObj_Mgr_enu/
start.swe?SWEExtSource=CustomUI&SWEExtCmd=Execute&WSSOAP=1
```

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

You control the session timeout length and session token timeout length and maximum age by setting the parameters listed in Table 13. These parameters are set in the [eai_/lang] section of the eapps.cfg file, which is located in the SWEAPP_ROOT\bin directory, where SWEAPP_ROOT is the directory in which you installed the SWSE.

Table 13. Session and Session Token Timeout-Related Parameters

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

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

For information on SessionTimeout, see Siebel Security Guide. For information on application configuration parameters in general, see Siebel System Administration Guide.
Examples of Using SOAP Headers for Authentication and Session Management

The following examples illustrate using Siebel Authentication and Session Management SOAP headers. These examples use various authentication and session type combinations. For more information, see "Combinations of Session Types and Authentication Types" on page 124.

Anonymous Request No Session
This example illustrates an anonymous request and a session type of None, which closes the session after the response is sent out:

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

Siebel Authorization No Session
This example illustrates a request that includes authentication credentials (username and password) and a session type of None, which closes the session after the response is sent out:

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

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

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

```xml
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <UsernameToken xmlns="http://siebel.com/webservices">user</UsernameToken>
    <PasswordText xmlns="http://siebel.com/webservices">hello123</PasswordText>
  </soap:Header>
  <soap:Body>
    <!-- data goes here -->
  </soap:Body>
</soap:Envelope>
```
Web Services ■ About Siebel Authentication and Session Management SOAP Headers

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

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:siebel-header="http://siebel.com/webservices"
    xmlns:SessionToken="http://siebel.com/webservices">
    <soap:Header>
        <SessionType xmlns="http://siebel.com/webservices">Stateless</SessionType>
        <SessionToken>2-r-JCunnMN9SxI9Any9zGQTOFlujEjfcXjfLoG-9Z00H4ljjbSd2P.7gyySzo07sFejxAU0WhdnK_</SessionToken>
    </soap:Header>
    <soap:Body>
        <!-- data goes here -->
    </soap:Body>
</soap:Envelope>

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

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:siebel-header="http://siebel.com/webservices"
    xmlns:SessionToken="http://siebel.com/webservices">
    <soap:Header>
        <SessionType xmlns="http://siebel.com/webservices">Stateless</SessionType>
        <SessionToken>2-r-JCunnMN9SxI9Any9zGQTOFlujEjfcXjfLoG-9Z00H4ljjbSd2P.7gyySzo07sFejxAU0WhdnK_</SessionToken>
    </soap:Header>
    <soap:Body>
        <!-- data goes here -->
    </soap:Body>
</soap:Envelope>

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

Initial Request
This example illustrates the initial request that includes authentication credentials (username and password) needed to log in:
About Siebel Authentication and Session Management SOAP Headers

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

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

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <siebel-header:SessionToken xmlns:siebel-header="http://siebel.com/webservices">Q7ABhvXBNUX5qTIoKJ9hZjhMzJ6lfTPa0oUDYxOBHkmOB7j</siebel-header:SessionToken>
  </soap:Header>
  <soap:Body>
    <!-- data goes here -->
  </soap:Body>
</soap:Envelope>

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

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <SessionType xmlns="http://siebel.com/webservices">Stateful</SessionType>
    <SessionToken xmlns="http://siebel.com/webservices">Q7ABhvXBNUX5qTIoKJ9hZjhMzJ6lfTPa0oUDYxOBHkmOB7j</SessionToken>
  </soap:Header>
  <soap:Body>
    <!-- data goes here -->
  </soap:Body>
</soap:Envelope>

Simple Query Starting With <soap:body>
This example illustrates data for a simple query starting with the <soap:body> element:

<soap:body>
  <Account_spcService_Account_spcServiceQueryPage_Input xmlns="http://siebel.com/CustomUI">
    <ListOfTestAccount
About Web Services and Web Single Sign-On Authentication

Siebel Web services support Web single sign-on (SSO) deployment scenarios in which third-party applications handle authentication, and then pass authentication information to the Siebel application. When the third-party application authenticates it, users do not have to explicitly log in to the Siebel application. Figure 30 illustrates a Web single SSO deployment scenario using Siebel Web services. For more information about Web SSO, see Siebel Security Guide.

Figure 30. Web Single Sign-On Scenario

Each component in the SSO scenario shown in Figure 30 is described in the following:

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

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

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

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

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

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

The following WSDL example, which shows named faults, is from http://www.gridlab.org:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MyService" targetNamespace="urn:myuri:1.0"
    xmlns:tns="urn:myuri:1.0"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:ns1="urn:myuri:1.0"
    xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/
    xmlns:MIME="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/">

<types>

</types>
</definitions>
```

Integration Platform Technologies: Siebel Enterprise Application Integration Version 8.1/8.2
Web Services □ About SOAP Fault Schema Support

```xml
<import namespace="http://schemas.xmlsoap.org/soap/encoding/">

<!-- fault element -->
<element name="MyFirstException">
  <complexType>
    <sequence>
      <element name="text" type="xsd:string" minOccurs="1" maxOccurs="1" nillable="false"/>
    </sequence>
  </complexType>
</element>

<!-- fault element -->
<element name="MySecondException">
  <complexType>
    <sequence>
      <element name="number" type="xsd:int" minOccurs="1" maxOccurs="1"/>
    </sequence>
  </complexType>
</element>

<!-- operation request element -->
<element name="myOperation">
  <complexType>
    <sequence>
      <element name="myInput" type="xsd:string" minOccurs="0" maxOccurs="1" nillable="true"/>
    </sequence>
  </complexType>
</element>

<!-- operation response element -->
<element name="myOperationResponse">
  <complexType>
    <sequence>
      <element name="myOutput" type="xsd:string" minOccurs="0" maxOccurs="1" nillable="true"/>
    </sequence>
  </complexType>
</element>

</schema>
</types>

<message name="myOperationRequest">
  <part name="parameters" element="ns1:myOperation"/>
</message>

<message name="myOperationResponse">
  <part name="parameters" element="ns1:myOperationResponse"/>
</message>

<message name="MyFirstExceptionFault">

```
Web Services

About SOAP Fault Schema Support

<Integration Platform Technologies: Siebel Enterprise Application Integration Version 8.1/8.2>

```xml
  <part name="fault" element="ns1:MyFirstException"/>
  </message>

  <message name="MySecondExceptionFault">
    <part name="fault" element="ns1:MySecondException"/>
  </message>

  <portType name="MyType">
    <operation name="myOperation">
      <documentation>Service definition of function ns1__myOperation</documentation>
      <input message="tns:myOperationRequest"/>
      <output message="tns:myOperationResponse"/>
      <fault name="MyFirstException" message="tns:MyFirstExceptionFault"/>
      <fault name="MySecondException" message="tns:MySecondExceptionFault"/>
    </operation>
  </portType>

  <binding name="MyService" type="tns:MyType">
    <operation name="myOperation">
      <SOAP:operation soapAction=""/>
      <input>
        <SOAP:body use="literal"/>
      </input>
      <output>
        <SOAP:body use="literal"/>
      </output>
      <fault name="MyFirstException">
        <SOAP:fault name="MyFirstException" use="literal"/>
      </fault>
      <fault name="MySecondException">
        <SOAP:fault name="MySecondException" use="literal"/>
      </fault>
    </operation>
  </binding>
```
The following SOAP message shows the first named fault from the example WSDL:

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

For more information on fault handling in Web services, see the SOAP and WSDL standards listed in “Supported Web Services Standards” on page 88.

## Handling SOAP Faults in Siebel CRM, Version 8.0 and Earlier

Siebel Business Applications version 8.0 and earlier ignore the schema for the named faults in a WSDL document. These SOAP faults are handled by returning an error code that indicated the occurrence of a SOAP fault. The data returned to the user is a property set with the following:
Three properties for the faultcode, faultstring, and faultactor subelements as output parameters
- A child of type XML Hierarchy with the application-specific errors from the detail subelement

Siebel Business Applications version 8.0 and earlier do not check for conformance to the original fault schema.

Handling SOAP Faults in Siebel Business Applications, Version 8.1 and Higher

In version 8.1 and higher, the fault schemas described in the WSDL are accepted and modeled as integration objects. This is similar to how other input and output messages are modeled as strings (simple type) or integration objects (complex type). These named faults are available as output parameters.

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

Handling SOAP Messages

SOAP messages are handled as follows:

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

Handling WSDL Imports

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

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

c If the Process Fault Schema check box is selected and an existing fault integration object is not specified, then a new fault integration object is added as an output parameter. The integration object name is prepended with Fault_.

For information on using the WSDL Import Wizard, see “Creating an Outbound Web Service Based on a WSDL File” on page 100.

### About Custom SOAP Filters

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

### Handling Custom Headers Using Filters

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

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

You must keep SOAP body and header processing isolated. The inbound dispatcher and outbound proxy can process the SOAP body, but cannot set or consume headers. Headers are application-specific. Some customization is needed to set and consume custom headers. To process optional custom headers that are provided by external applications, a special business service, a filter, is defined. You can configure the Web service outbound proxy and the Web service inbound dispatcher to call specific filters for the processing of individual (custom) headers.

**NOTE:** The SOAP header will not be passed to the underlying business service or workflow of the inbound Web service. Any processing that must be done with the SOAP header must be done on the filter business service.

### Enabling SOAP Header Processing Through Filters

For each operation, you can set the inbound and outbound filters to be run. You can also define the methods you want to call on the filter.
The following code sample illustrates a filter that has been written for the handling of custom SOAP headers. The interface provided by this code sample lets you define the method on the filter that you want to call, as well as the corresponding input and output parameters.

```javascript
function Service_PreInvokeMethod (MethodName, Inputs, Outputs) {
  if(MethodName == 'StripHeader') {
    if(Inputs.GetChildCount() > 0) {
      // Set the input SOAP message property set as the output.
      Outputs.InsertChildAt(Inputs.GetChild(0), 0);
      var soapEnv = Outputs.GetChild(0);
      if(soapEnv.GetChildCount() == 2) // headers and body {
        // Here is where the header is found and processed.
        var count = soapEnv.GetChildCount();
        var i = 0;
        var headerParam1;
        var headerParam2;
        var headerParam3;
        // Use a loop just in case the header is not the first hierarchy.
        for (; i < count; i++) {
          // For simplicity, the string comparison must be done using the exact same value
          // as the SOAP message tag name.
          if (soapEnv.GetChild(i).GetType() == "soapenv:Header") {
            // Found the header. Now it is processed.
            var soapHeader = soapEnv.GetChild(i);
            // This example assumes that the following header hierarchy is received:
            // <soapEnv:Header>
            //   <headerParam1>Value1</headerParam1>
            //   <headerParam2>Value2</headerParam2>
            //   <headerParam3>Value3</headerParam3>
            // </soapEnv:Header>
            // The parameters headerParam1, headerParam2, and headerParam3
            // are saved into variables. Nothing further done with them.
```
headerParam1 = soapHeader.GetChild(0);
headerParam2 = soapHeader.GetChild(1);
headerParam3 = soapHeader.GetChild(2);
break; // Stop the loop after the header is found.
}

// Must remove the header from the SOAP property set.
soapEnv.RemoveChild(i);
}
}

else if(MethodName == "AddHeader") {

if(Inputs.GetChildCount() > 0) {
    // Create the SOAP header hierarchy with the desired data.
    var soapHeader = TheApplication().NewPropertySet();
    soapHeader.SetType("soapEnv:Header");
    soapHeader.SetProperty("xmlns:soapEnv",
                              "http://schemas.xmlsoap.org/soap/envelope/");
    // These will be created as property sets because we want the following header:
    // <soapEnv:Header>
    //     <headerParam1>Value1</headerParam1>
    //     <headerParam2>Value2</headerParam2>
    //     <headerParam3>Value3</headerParam3>
    // </soapEnv:Header>
    var param1PS = TheApplication().NewPropertySet();
    var param2PS = TheApplication().NewPropertySet();
    var param3PS = TheApplication().NewPropertySet();
    param1PS.SetType("headerParam1");
    param1PS.SetValue("Value1");
    param2PS.SetType("headerParam2");
    param2PS.SetValue("Value2");
    param3PS.SetType("headerParam3");
    param3PS.SetValue("Value3");
    soapHeader.AddChild(param1PS);
    soapHeader.AddChild(param2PS);
    soapHeader.AddChild(param3PS);
    soapEnv.AddChild(soapHeader);
}
}
param2PS.SetValue("Value2");
param3PS.SetType("headerParam3");
param3PS.SetValue("Value3");
// Add the data to the SOAP header.
soapHeader.AddChild(param1PS);
soapHeader.AddChild(param2PS);
soapHeader.AddChild(param3PS);
// Get the SOAP envelope from the SOAP hierarchy.
var soapEnv = Inputs.GetChild(0);
// Add the header to the SOAP envelope.
soapEnv.InsertChildAt(soapHeader, 0);
Outputs.InsertChildAt(soapEnv, 0);
}
}
return(CancelOperation);

**Inputting a SOAP Envelope to a Filter Service**

Using a SOAP envelope as the input to a filter service is the property set representation of an XML document. For example, each tag in the XML document is a property set. Each attribute on the tag is a property in the property set.

To pass the information in the headers further down the stack to the actual business service method or workflow being called, the *HeaderContext* property set is passed to the business service or workflow that is called. For example, on a call to an inbound Web service, if there are a couple of headers in the SOAP message, the filter service extracts the header information. To use this information in the business service or workflow execution call, it has to be contained in the *HeaderContext*. Internally, the Siebel Web services infrastructure passes *HeaderContext* to the eventual business service or workflow that is called.
About EAI File Streaming

Siebel Business Applications support streaming of EAI requests and responses encountered. This feature allows the Siebel Web Engine (SWE) and the EAI Object Manager (OM) to process Web service calls that involve large requests or responses. Large requests and responses can occur when inserting or querying file attachments by way of a Web service. By transferring data internally in 100-KB chunks, the memory footprints of the Siebel Web Engine and EAI Object Manager processes are reduced and system scalability is improved.

This topic describes the streaming process for inbound EAI requests and outbound responses.

Enabling the EAI Streaming Process

To enable EAI streaming, you must first set the LargeFileChunkingEnabled parameter to TRUE in the eapps.cfg file.

About Inbound EAI Streaming Requests

Figure 31 provides an overview of the components and process flow used for streaming an inbound request.

Figure 31 illustrates the following process:

1. When the Siebel Web Server Extension (SWSE) Plug-in receives the request from the HTTP client by way of the Web Server, it determines that this request must be chunked based on the HTTP request body size, embeds streaming information in the request, and then sends the first chunk of the request body to the SWE Server.

2. The SWE Server extracts the streaming information from the request, determines it is a streaming request, and then writes the first chunk to the file system and sends a response to the plug-in indicating that the request has been processed. The SWSE Plug-in sends the next chunk and the cycle continues until the last chunk has been written to the file system.
3 After the entire body of the HTTP request has been written to a file on the disk, the SWE Server calls the Web service method on EAI Object Manager, passing the name of the file as an input argument.

**About Outbound EAI Streaming Responses**

*Figure 32* provides an overview of the components and process flow used for streaming an outbound response.

![Image](image.png)

*Figure 32. Request-Response Cycle(s) for an Outbound Streaming Response*

*Figure 32* illustrates the following process:

1 An EAI request from the HTTP client is received by the SWSE Plug-in.

2 The SWSE Plug-in forwards the request to the SWE Server.

3 The SWE Server then performs one of the following actions:
   a If the request is not a streaming request, the SWE Server calls the EAI Object Manager method.
   b If the request is a streaming request, then the file is first written to disk before the SWE Server calls the EAI Object Manager method.

4 The EAI Object Manager forwards the response to the SWE Server, and SWE queries the output arguments.

5 If a file reference is found, then the SWE Server transmits the file to the client.

   Whether multiple chunks will be sent or not depends upon the size of the file. If chunking is needed, then the SWE Server sends the first chunk to the plug-in, also embedding the streaming information in the response.

6 The Plug-in sends the chunk to the client including the HTTP headers in the response, and then it requires the SWE Server to get the next chunk.

   **NOTE:** The plug-in sends the HTTP response headers only for the first chunk.

The cycle continues until the entire file has been transmitted to the HTTP client.
**About EAI Streaming Criteria**

The following criteria are used internally to decide whether a particular request or response is streamed:

1. The Web Server Plug-in initiates chunking of an inbound EAI request only if the following conditions are met:
   - a. The EnableExtServiceOnly parameter is set to TRUE in the [Application] section of eapps.cfg.
   - b. A new parameter named LargeFileChunkingEnabled is explicitly set to TRUE in the [Application] section of eapps.cfg.
   - c. The size of the body of the inbound HTTP request is greater than 100 KB.

   If these conditions are not met, then the plug-in does not stream the contents of the EAI request, and the request is processed as one chunk.

2. The SWE server initiates outbound chunking only if the following conditions are met:
   - a. The SWE finds a property named ExtSvcFileName in the output arguments after calling the EAI Object Manager method.
      - The value of this property must be a fully qualified path, and the name of the response file is written to disk by EAI.
   - b. The file size is greater than 100 KB.

   If these conditions are not met, then the plug-in does not stream the contents of the EAI response, and the request is processed as one chunk.

**About Web Services Cache Refresh**

Both Siebel inbound and outbound Web services are typically cached into memory on the Siebel Server. At times, administrators must update the definitions of these services to provide more current or correct functionality. Administrators have the ability to directly refresh the memory cache in real time, without stopping and restarting the Siebel server.

The Web services cache is used to store all the global administration information that can be manipulated in the Inbound and Outbound Web Service administration views.

The Clear Cache feature is a button on the Administration - Web Services screen. This feature is available for inbound and outbound Web services. When the administrator decides when the Web service configuration must be refreshed, he or she clicks Clear Cache.

When Clear Cache is clicked, the integration object and Web services definitions in the run-time database are invalidated. Object definitions are reloaded when requested in the client.
Enabling Web Services Tracing

You can enable Web services tracing on the Siebel Server to write all inbound and outbound SOAP documents to a log file.

To enable Web services tracing

1. Navigate to the Administration - Server Configuration screen, Servers view.
   The view that appears displays three different list applets. The top applet lists the Siebel Servers for the enterprise. The middle applet has three tabs: Components, Parameters, and Events. The bottom applet has two tabs: Events and Parameters.

2. In the top list applet, select the Siebel Server that you want to configure.

3. In the middle applet, click the Components tab.
   This list applet contains the components for the Siebel Server selected in the top applet.
   Choose the relevant application object manager.

4. In the bottom applet, click the Parameters tab.
   This list applet contains the parameters for the Component selected in the middle applet.

5. Set the Log Level to 4 for any or all of the following Event Types.

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Alias</th>
<th>Description</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Service Performance</td>
<td>WebSvcPerf</td>
<td>Web Service Performance Event Type</td>
<td>Used for performance logging</td>
</tr>
<tr>
<td>Web Service Outbound Argument Tracing</td>
<td>WebSvcOutboundArgTrc</td>
<td>Web Service Outbound Run-time Argument Tracing</td>
<td>Used for logging arguments to the outbound dispatcher</td>
</tr>
<tr>
<td>Web Service Outbound</td>
<td>WebSvcOutbound</td>
<td>Web Service Outbound Run-time Event Type</td>
<td>Used for run-time logging of outbound Web services</td>
</tr>
<tr>
<td>Web Service Loading</td>
<td>WebSvcLoad</td>
<td>Web Service Configuration Loading Event Type</td>
<td>Used for logging of the loading of Web services</td>
</tr>
<tr>
<td>Web Service Inbound Argument Tracing</td>
<td>WebSvcInboundArgTrc</td>
<td>Web Service Inbound Run-time Argument Tracing</td>
<td>Used for logging arguments to the inbound dispatcher</td>
</tr>
</tbody>
</table>
6 In the middle applet, click the Components tab.

7 Select the EAI Object Manager component, and then click the Parameters tab.
   The Component Parameters list appears.

8 Click Advanced to see the advanced parameters. (Click Reset to hide them again.)

9 Query for Enable Business Service Argument Tracing.

10 Set its Value and Value on Restart fields to True.

11 Restart or reconfigure the server component.
   For information on restarting server components and on advanced and hidden parameters, see
   *Siebel System Administration Guide*. 

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Alias</th>
<th>Description</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Service Inbound</td>
<td>WebSvcInbound</td>
<td>Web Service Inbound Run-time Event Type</td>
<td>Used for logging at Web service inbound run time. Information is logged to the inbound dispatcher</td>
</tr>
<tr>
<td>Web Service Design</td>
<td>WebSvcDesign</td>
<td>Web Service Design-time Event Type</td>
<td>Used for logging at Web service design time. For example, at the time of WSDL import and generation</td>
</tr>
</tbody>
</table>
EAI Siebel Adapter is a preconfigured business service that is used with any integration process that runs through the Siebel business object layer. Integration objects are used to update data in business objects and are used when retrieving data from business objects. These integration objects are configurable and can be used during an integration process (for example, entering and retrieving data from the Siebel Business Application).

This chapter describes the functionality of the EAI Siebel Adapter business service, and the different methods and arguments you can use with it to manipulate the data in the Siebel Database. It includes:

- About the EAI Siebel Adapter Business Service on page 145
- EAI Siebel Adapter Business Service Methods on page 146
- EAI Siebel Adapter Business Service Method Arguments on page 171
- About Multivalue Groups in the EAI Siebel Adapter Business Service on page 175
- About Using Effective Dating with Siebel EAI Adapter Business Service on page 176
- Enabling Effective Dating on Fields on page 177
- Enabling Effective Dating on Links on page 181
- About Using Language-Independent Code with the EAI Siebel Adapter Business Service on page 185
- Siebel EAI and Run-Time Events on page 186
- Guidelines for Using the EAI Siebel Adapter Business Service on page 187
- Troubleshooting the EAI Siebel Adapter Business Service on page 188
- Enabling Logging for the EAI Siebel Adapter Business Service on page 188
- Enabling Siebel Argument Tracing on page 190
- Configuring the EAI Siebel Adapter Business Service for Concurrency Control on page 191

About the EAI Siebel Adapter Business Service

EAI Siebel Adapter is a general-purpose integration business service that allows you to:

- Read Siebel business objects from the Siebel Database into integration objects.

**NOTE:** When called locally, the EAI Siebel Adapter business service creates an additional database connection. If this second connection times out, then further transactions will not be processed. To prevent this from happening, add the TrxDBConnReconnectIntervalSeconds parameter to the [ServerDataSrc] section of the application configuration (CFG) file. A typical value is 120.
Write an integration object instance whose data originates externally in a Siebel business object.

Update multiple corresponding top-level parent business component records with data from one XML file, for an example, see “Update Method” on page 159.

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

---

**Node Types and the EAI Siebel Adapter Business Service**

In an integration object hierarchy, nodes with at least one child are called internal nodes and nodes without children are called leaf nodes. When either the insert or update method is called on the EAI Siebel Adapter business service, the adapter performs the operation on both internal nodes and leaf nodes. When the insert or update method is called on the EAI UI Data Adapter business service, the adapter performs insert on leaf nodes only.

For more information on node types, see “About the EAI UI Data Adapter Business Service” on page 197.

---

**EAI Siebel Adapter Business Service Methods**

The EAI Siebel Adapter supports the following methods:

- “Query Method” on page 147
- “QueryPage Method” on page 148
- “Synchronize Method” on page 150
- “Insert Method” on page 157
- “Upsert Method” on page 159
- “Update Method” on page 159
- “Delete Method” on page 160
- “Execute Method” on page 160

---

**About the Examples in the EAI Siebel Adapter Business Service Methods Sections**

The following information is true for the examples used for the EAI Siebel Adapter methods:

- The business object data is represented as integration object data in XML format.
- The XML document or integration object instance might also be referred to as a Siebel Message.
Fields that contain null values are not included in the XML examples. However, these fields might be revealed when you use EAI XML Write to File.WriteEAIMsg() to print out the XML.

**Query Method**

You can use a combination of input arguments when using the Query Business Service Method of the EAI Siebel Adapter. The input arguments are as follows:

1. **Query By Example (QBE).** Pass in an integration object instance represented as a property set. 
   
   **NOTE:** EAI queries on integration objects do not use a search specification for child integration components when the query obtains the parent integration component.

2. **Primary Row Id.** Pass in a string to the Object Id input argument. The string can be the row_id of the primary business component of the Output Integration Object Name.

3. **Output Integration Object Name.** See the Primary Row Id for information.

4. **Search Specification.** Pass in a String expression.

The input arguments can be used in one of the following combinations:

- 1
- 2 and 3
- 4
- 3 and 4
- 2, 3, and 4

The EAI Siebel Adapter uses this input as criteria to query the base business object and to return a corresponding integration object instance.

For an example of using the search specification method argument to limit the scope of your query see "About Using Language-Independent Code with the EAI Siebel Adapter Business Service" on page 185.

When using the EAI Siebel Adapter, to query all the business component records, you are not required to specify any value in the Object Id process property of the workflow. In this case, not specifying an Object Id or a Search Specification works as a wildcard.

If you want to query Siebel data using the EAI Siebel Adapter with the Query method and an integration object instance (property set) containing a query by example (QBE) search criterion, then all the fields present in the QBE will be used in the query. To retrieve a unique record, include the fields that make up the user key for the underlying integration object component instance to ensure you retrieve a unique record. You can use an asterisk (*) as a wildcard for each one of the fields.

For example, the following is your QBE:

```xml
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
```
<SiebelMessage MessageId = "1-210Y" IntObjectName = "EAI Account" MessageType = "Integration Object" IntObjectFormat = "Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <CSN>*</CSN>
      <HomePage>*</HomePage>
      <Location>H*</Location>
      <Name>A*</Name>
      <Type>*</Type>
    </Account>
  </ListOfAccount>
</SiebelMessage>

You would receive all of the Accounts with names that start with A* and have locations that start with H*. The CSN, HomePage, and Type fields cannot be blank because they are used in the query.

The EAI Siebel Adapter converts the QBE into a user Search Expression of the following:

```
```

You can run this example and review the output XML generated.

When using search expressions that contain an apostrophe ('), you must use two apostrophes ('' ) or the search will fail.

For example, you are searching for the string LUKE'S. The following search expression will fail in the EAI Siebel Adapter:

```
[Account.Name] LIKE "*LUKE'S*"
```

Use the following search expression instead:

```
[Account.Name] LIKE "*LUKE'S*"
```

**NOTE:** The EAI Siebel Adapter explicitly overrides any Object Manager settings for the MaxCursorSize parameter. The EAI Siebel Adapter uses a MaxCursorSize of -1. If you want to limit the number of results received when using the Query method, then use the QueryPage Method. You can combine the Object Id and Search Specification together to query for parent and child data.

**NOTE:** The EAI Siebel Adapter returns the output of the Query() method as one Siebel Message. This integration object instance is stored in the process memory. If your query returns a large number of records, this will result in your Siebel component's memory consumption being high.

**QueryPage Method**

This method is useful when the search specification retrieves a large number of records at the root component. To avoid returning one huge Siebel Message, you can specify the number of records to be returned using the PageSize argument, as presented in Table 25 on page 172. You can also use method arguments such as OutputIntObjectName, SearchSpec, SortSpec, ViewMode, and StartRowNum to dictate which records to return.
Even though the QueryPage returns a limited number of records, it keeps the data in the cache, which you can then retrieve by calling the EAI Siebel Adapter with a new value for the StartRowNum method argument. 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.

**NOTE:** The EAI Siebel Adapter returns the output of the QueryPage() method as one Siebel Message. This integration object instance is stored in the process memory. If your query returns a large number of records, this will result in your Siebel component's memory consumption being high.

The QueryPage method precedes each integration object instance. It is provided through the SiebelMessage input argument when performing a query by example. Parameters such as StartRowNum, PageSize, and others are applied to each integration object instance.

For example, a Siebel database contains four account records with the Name field set to: a1, a2, b1, b2. An input SiebelMessage with two instances of the Account integration object with the first instance's name set to "a*" and the second instance's name set to "b*" the result for StartRowNum=0 is all four records (a1, a2, b2, b4), and for StartRowNum=1 only two records (a2 and b2). This example illustrates that the StartRowNum method argument counts records within each single Integration Object instance of the query by example input SiebelMessage: once for "a*" (a1, a2) and once for "b*" (b1, b2).

**Business Service Example**

The following is an example of using the QueryPage() method in a business service.

```javascript
var EAIService = TheApplication().GetService("EAI Siebel Adapter");
var writeSvc = TheApplication().GetService("EAI XML Write to File");
var EAILin = TheApplication().NewPropertySet();
var ResultSet = TheApplication().NewPropertySet();
var moreRecords = true;
var countOfObjects = 0;
var i = 1;

// set up input arguments, get 10 at a time
EAILin.SetProperty("OutputIntObjectName", "EAI Account");
EAILin.SetProperty("PageSize", "10");
EAILin.SetProperty("SearchSpec", ":[Account.Name] LIKE '3*'");
EAILin.SetProperty("StartRowNum", i);
EAILin.SetProperty("NewQuery", "true");

// retrieve the business component data
EAIService.InvokeMethod("QueryPage", EAILin, ResultSet);

// loop through cached data
while ( (ResultSet.GetChildCount() > 0) && (moreRecords) ) {
    countOfObjects = countOfObjects + ResultSet.GetProperty("NumOutputObjects");
    // write out first chunk of data retrieved
    ResultSet.SetProperty("FileName", "d:\temp\EAIaccount$$.xml");
    writeSvc.InvokeMethod("WriteEAIMsg", ResultSet, Outputs);
}

// reuse the existing input property set, except don't reissue query
```
EAI in SetProperty("NewQuery", "false");
i = i+10; // get next 10 records
EAI in SetProperty("StartRowNum", i);

ResultSet.Reset(); // clear previous result set
EAI Service.InvokeMethod("QueryPage", EAI in, ResultSet);

if (ResultSet.GetProperty("LastPage") == "true")
    moreRecords = false;
}

Synchronize Method

You can use the Synchronize method to make the values in a business object instance match those of an integration object instance. This operation can result in updates, insertions, or deletions in the business components. The following rules apply to the results of this method:

- If a child component is not present in the integration object instance, then the corresponding child business component rows are left untouched.
- If the integration object instance’s child component has an empty container, then all child records in the corresponding business component are deleted.
- For a particular child component, records that exist in both the integration object instance and business component are updated. Records that exist in the integration object hierarchy and not in the business component are inserted. Records in the business component and not in the integration object instance are deleted.
- Only the fields specified in the integration component instance are updated.

**NOTE:** When the EAI Siebel Adapter starts a database transaction (initiated to allow updates to the Siebel database) it must ensure the data queried is committed and consistent. The results of these queries dictate what changes are applied, so if reads that contain uncommitted data (*dirty reads*) were enabled, it could cause incorrect updates by the EAI Siebel Adapter. Therefore, dirty reads are disabled during database transactions started by the EAI Siebel Adapter.

Example of Synchronize Method on Deleted Unmatched Children

This first example demonstrates deleting unmatched children when using the Synchronize method. This example uses data present in the sample database.

```xml
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QYS" IntObjectName = "EAI Account" MessageType = "Integration Object" IntObjectFormat = "Siebel Hierarchical">
    <ListOfAccount>
        <Account>
            <AccountStatus>Active</AccountStatus>
            <CSN>1-3JGO7</CSN>
            <CurrencyCode>USD</CurrencyCode>
            <LanguageCode>ENU</LanguageCode>
        </Account>
    </ListOfAccount>
</SiebelMessage>
```
<Location>Test</Location>
<Name>ABC Corp</Name>
<ListOfAccount_BusinessAddress>
  <Account_BusinessAddress IsPrimaryMVG = "Y">
    <AddressActiveStatus>Y</AddressActiveStatus>
    <BillAddressFlag>Y</BillAddressFlag>
    <City>ATown</City>
    <Country>USA</Country>
    <MainAddressFlag>Y</MainAddressFlag>
    <ShipAddressFlag>Y</ShipAddressFlag>
    <StreetAddress>123 Main St</StreetAddress>
  </Account_BusinessAddress>
  <Account_BusinessAddress IsPrimaryMVG = "N">
    <AddressActiveStatus>Y</AddressActiveStatus>
    <BillAddressFlag>Y</BillAddressFlag>
    <City>BTown</City>
    <Country>USA</Country>
    <MainAddressFlag>Y</MainAddressFlag>
    <ShipAddressFlag>Y</ShipAddressFlag>
    <StreetAddress>456 Oak St</StreetAddress>
  </Account_BusinessAddress>
</ListOfAccount_BusinessAddress>
<ListOfContact>
  <Contact>
    <ActiveStatus>Y</ActiveStatus>
    <FirstName>User1</FirstName>
    <LastName>User1</LastName>
    <Organization>Default Organization</Organization>
    <ListOfContact_Organization>
      <Contact_Organization IsPrimaryMVG = "Y">
        <Organization>Default Organization</Organization>
        <OrganizationIntegrationId/>
      </Contact_Organization>
    </ListOfContact_Organization>
    <ListOfContact_AlternatePhone/>
  </Contact>
  <Contact>
    <ActiveStatus>Y</ActiveStatus>
    <FirstName>User2</FirstName>
    <LastName>User2</LastName>
    <Organization>Default Organization</Organization>
    <ListOfContact_Organization>
      <Contact_Organization IsPrimaryMVG = "Y">
        <Organization>Default Organization</Organization>
        <OrganizationIntegrationId/>
      </Contact_Organization>
    </ListOfContact_Organization>
    <ListOfContact_AlternatePhone/>
  </Contact>
</ListOfContact>
<ListOfAccount_Organization>
  <Account_Organization IsPrimaryMVG = "Y">
    <Organization>Default Organization</Organization>
    <OrganizationIntegrationId>0-R9NH</OrganizationIntegrationId>
  </Account_Organization>
</ListOfAccount_Organization>
Integration Platform Technologies: Siebel Enterprise Application Integration Version 8.1/8.2

EAI Siebel Adapter Business Service

Then the following XML (integration object instance) is submitted with Synchronize:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId="1-2QY5" IntObjectName="EAI Account" MessageType="Integration Object" IntObjectFormat="Siebel Hierarchical">
    <ListOfAccount>
        <Account>
            <AccountStatus>Active</AccountStatus>
            <CSN>1-3JGO7</CSN>
            <Competitor>Y</Competitor>
            <CurrencyCode>USD</CurrencyCode>
            <LanguageCode>CHS</LanguageCode>
            <Location>test</Location>
            <Name>ABC Corp</Name>
            <ListOfContact>
                <Contact>
                    <ActiveStatus>N</ActiveStatus>
                    <FirstName>User1</FirstName>
                    <LastName>User1</LastName>
                    <Organization>Default Organization</Organization>
                </Contact>
                <Contact>
                    <FirstName>User3</FirstName>
                    <LastName>User3</LastName>
                    <Organization>Default Organization</Organization>
                </Contact>
            </ListOfContact>
        </Account>
    </ListOfAccount>
</SiebelMessage>
```

Following is the result you will receive. Because the contact information is included in the integration object instance, User2 in the database is deleted because it was an unmatched node. User1 is updated because it is a matched node. User3 is inserted because it is a new node. Since Business Address was not included in the integration object instance, it is left in the business object.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId="1-2QY5" IntObjectName="EAI Account" MessageType="Integration Object" IntObjectFormat="Siebel Hierarchical">
    <ListOfAccount>
        <Account>
            <AccountStatus>Active</AccountStatus>
            <CSN>1-3JGO7</CSN>
        </Account>
    </ListOfAccount>
</SiebelMessage>
```
<CurrencyCode>USD</CurrencyCode>
<LanguageCode>CHS</LanguageCode>
<Location>Test</Location>
>Name>ABC Corp</Name>
<ListOfAccount_BusinessAddress>
  <Account_BusinessAddress IsPrimaryMVG = "Y">
    <AddressActiveStatus>Y</AddressActiveStatus>
    <BillAddressFlag>Y</BillAddressFlag>
    <City>ATown</City>
    <Country>USA</Country>
    <MainAddressFlag>Y</MainAddressFlag>
    <ShipAddressFlag>Y</ShipAddressFlag>
    <StreetAddress>123 Main St</StreetAddress>
  </Account_BusinessAddress>
  <Account_BusinessAddress IsPrimaryMVG = "N">
    <AddressActiveStatus>Y</AddressActiveStatus>
    <BillAddressFlag>Y</BillAddressFlag>
    <City>BTown</City>
    <Country>USA</Country>
    <MainAddressFlag>Y</MainAddressFlag>
    <ShipAddressFlag>Y</ShipAddressFlag>
    <StreetAddress>456 Oak St</StreetAddress>
  </Account_BusinessAddress>
</ListOfAccount_BusinessAddress>
<ListOfContact>
  <Contact>
    <ActiveStatus>N</ActiveStatus>
    <FirstName>User1</FirstName>
    <LastName>User1</LastName>
    <Organization>Default Organization</Organization>
    <ListOfContact_Organization>
      <Contact_Organization IsPrimaryMVG = "Y">
        <Organization>Default Organization</Organization>
      </Contact_Organization>
    </ListOfContact_Organization>
  </Contact>
  <Contact>
    <ActiveStatus>N</ActiveStatus>
    <FirstName>User3</FirstName>
    <LastName>User3</LastName>
    <Organization>Default Organization</Organization>
    <ListOfContact_Organization>
      <Contact_Organization IsPrimaryMVG = "Y">
        <Organization>Default Organization</Organization>
      </Contact_Organization>
    </ListOfContact_Organization>
  </Contact>
</ListOfContact>
</Account>
</ListOfAccount>
Table 14 is a high-level representation of the previous example.

<table>
<thead>
<tr>
<th>Record In Database</th>
<th>Integration Object Instance</th>
<th>Record After Synchronize</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account: ABC Corp</td>
<td>Account: ABC Corp</td>
<td>Account: ABC Corp</td>
</tr>
<tr>
<td>Business Address: 123 Main St</td>
<td>Contact: User1</td>
<td>Business Address: 123 Main St</td>
</tr>
<tr>
<td>Business Address: 456 Oak St</td>
<td>Contact: User3</td>
<td>Business Address: 456 Oak St</td>
</tr>
<tr>
<td>Contact: User1</td>
<td>Organization: Default Org.</td>
<td>Contact: User1</td>
</tr>
<tr>
<td>Contact: User2</td>
<td>Organization: Default Org.</td>
<td>Contact: User3</td>
</tr>
</tbody>
</table>

**Effect of the Synchronize Method on Records with Empty Containers**

This second example demonstrates how all records with an empty container are deleted when using the Synchronize method.

If you start with this business component data:

```xml
<?xml version = "1.0" encoding = "UTF-8"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType = "Integration Object" IntObjectFormat = "Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <AccountStatus>Active</AccountStatus>
      <CSN>1-3J GO7</CSN>
      <CurrencyCode>USD</CurrencyCode>
      <LanguageCode>ENU</LanguageCode>
      <Location>test</Location>
      <Name>ABC Corp</Name>
      <ListOfAccount_BusinessAddress>
        <Account_BusinessAddress IsPrimaryMVG = "Y">
          <AddressId>1-3J GOA</AddressId>
          <AddressActiveStatus>Y</AddressActiveStatus>
          <BillAddressFlag>Y</BillAddressFlag>
          <City>MyTown</City>
          <Country>Canada</Country>
          <MainAddressFlag>Y</MainAddressFlag>
          <ShipAddressFlag>Y</ShipAddressFlag>
          <StreetAddress>123 Main St</StreetAddress>
        </Account_BusinessAddress>
        <Account_BusinessAddress IsPrimaryMVG = "N">
          <AddressId>1-3J GOA</AddressId>
          <AddressActiveStatus>Y</AddressActiveStatus>
          <BillAddressFlag>Y</BillAddressFlag>
          <City>MyTown</City>
          <Country>Canada</Country>
          <MainAddressFlag>Y</MainAddressFlag>
          <ShipAddressFlag>Y</ShipAddressFlag>
          <StreetAddress>456 Oak St</StreetAddress>
        </Account_BusinessAddress>
        <Account_BusinessAddress IsPrimaryMVG = "Y">
          <AddressId>1-3J GOA</AddressId>
          <AddressActiveStatus>Y</AddressActiveStatus>
          <BillAddressFlag>Y</BillAddressFlag>
          <City>MyTown</City>
          <Country>Canada</Country>
          <MainAddressFlag>Y</MainAddressFlag>
          <ShipAddressFlag>Y</ShipAddressFlag>
          <StreetAddress>123 Main St</StreetAddress>
        </Account_BusinessAddress>
      </ListOfAccount_BusinessAddress>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```
And the following integration object instance is passed in:

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

<AddressActiveStatus>Y</AddressActiveStatus>
<BillAddressFlag>Y</BillAddressFlag>
<AddressId>1-3 JG0B</AddressId>
<City>YourTown</City>
<Country>Canada</Country>
<MainAddressFlag>Y</MainAddressFlag>
<ShipAddressFlag>Y</ShipAddressFlag>
<StreetAddress>456 Oak St</StreetAddress>
</Account_BusinessAddress>
</ListOfAccount_BusinessAddress>
<Contact>
  <ActiveStatus>Y</ActiveStatus>
  <FirstName>User1</FirstName>
  <LastName>User1</LastName>
  <MiddleName/>
  <Organization>Default Organization</Organization>
  <ListOfContact_Organization>
    <Contact_Organization IsPrimaryMVG = "Y">
      <Organization>Default Organization</Organization>
    </Contact_Organization>
  </ListOfContact_Organization>
  <ListOfContact_AlternatePhone/>
</Contact>
<Contact>
  <ActiveStatus>Y</ActiveStatus>
  <FirstName>User2</FirstName>
  <LastName>User2</LastName>
  <MiddleName/>
  <Organization>Default Organization</Organization>
  <ListOfContact_Organization>
    <Contact_Organization IsPrimaryMVG = "Y">
      <Organization>Default Organization</Organization>
    </Contact_Organization>
  </ListOfContact_Organization>
  <ListOfContact_AlternatePhone/>
</Contact>
</ListOfContact>
<ListOfAccount_Organization>
  <Account_Organization IsPrimaryMVG = "Y">
    <Organization>Default Organization</Organization>
    <OrganizationId>0-R9NH</OrganizationId>
  </Account_Organization>
</ListOfAccount_Organization>
</Account>
</ListOfAccount>
</SiebelMessage>
```
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType = "Integration Object" IntObjectFormat = "Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <AccountStatus>Active</AccountStatus>
      <CSN>1-3JGO7</CSN>
      <Competitor>Y</Competitor>
      <CurrencyCode>USD</CurrencyCode>
      <LanguageCode>CHS</LanguageCode>
      <Location>test</Location>
      <Name>ABC Corp</Name>
      <ListOfContact/>
    </Account>
  </ListOfAccount>
</SiebelMessage>

After the sync operation, all the children contacts are deleted because none of the nodes match.

<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType = "Integration Object" IntObjectFormat = "Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <AccountStatus>Active</AccountStatus>
      <CSN>1-3JGO7</CSN>
      <CurrencyCode>USD</CurrencyCode>
      <LanguageCode>ENU</LanguageCode>
      <Location>test</Location>
      <Name>ABC Corp</Name>
      <ListOfAccount_BusinessAddress>
        <Account_BusinessAddress IsPrimaryMVG = "Y">
          <AddressId>1-3JGOA</AddressId>
          <AddressActiveStatus>Y</AddressActiveStatus>
          <BillingAddressFlag>Y</BillingAddressFlag>
          <City>MyTown</City>
          <Country>Canada</Country>
          <MainAddressFlag>Y</MainAddressFlag>
          <ShipAddressFlag>Y</ShipAddressFlag>
          <StreetAddress>123 Main St</StreetAddress>
        </Account_BusinessAddress>
        <Account_BusinessAddress IsPrimaryMVG = "N">
          <AddressId>1-3JGOB</AddressId>
          <AddressActiveStatus>Y</AddressActiveStatus>
          <BillingAddressFlag>Y</BillingAddressFlag>
          <City>YourTown</City>
          <Country>Canada</Country>
          <MainAddressFlag>Y</MainAddressFlag>
          <ShipAddressFlag>Y</ShipAddressFlag>
          <StreetAddress>456 Oak St</StreetAddress>
        </Account_BusinessAddress>
      </ListOfAccount_BusinessAddress>
    </Account>
  </ListOfAccount>
</SiebelMessage>
Table 15 is a high level representation of the operation.

Table 15. Representation of Records with Empty Containers Being Deleted Using Synchronize Method

<table>
<thead>
<tr>
<th>Record In Database</th>
<th>Integration Object Instance</th>
<th>Record After Synchronize Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account: ABC Corp</td>
<td>Account: ABC Corp</td>
<td>Account: ABC Corp</td>
</tr>
<tr>
<td>■ Business Address: 123 Main St</td>
<td>■ Contact:</td>
<td>■ Business Address: 123 Main St</td>
</tr>
<tr>
<td>■ Business Address: 456 Oak St</td>
<td>■</td>
<td>■ Business Address: 456 Oak St</td>
</tr>
</tbody>
</table>

Insert Method

This method is also similar to the Synchronize method with the exception that the EAI Siebel Adapter generates an error if a matching root component is found; otherwise, it inserts the root component and synchronizes all the children. It is important to note that when you insert a record, there is a possibility that the business component would create default children for the record, which will be removed by the Insert method. The Insert method synchronizes the children, which deletes all the default children. For example, if you insert an account associated with a specific organization, then it will also be automatically associated with a default organization. As part of the Insert method, the EAI Siebel Adapter deletes the default association, and associates the new account with only the organization that was originally defined in the input integration object instance. The EAI Siebel Adapter achieves this by synchronizing the children.

Example of Using the Insert Method

If you use the Insert method with the example of the integration object instance represented in XML that follows, then a new service request is created with two activities.
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2R6E" IntObjectName = "Sample Service Request" MessageType = "Integration Object" IntObjectFormat = "Siebel Hierarchical">
<ListOfSampleServiceRequest>
<ServiceRequest>
<Account>Genesys Communications</Account>
<AccountLocation>San Francisco, CA</AccountLocation>
/Area>Network</Area>
<ClosedDate/>
<CommitTime/>
<ContactBusinessPhone>4155551100</ContactBusinessPhone>
<ContactLastName>Kastrup-Larsen</ContactLastName>
>Description>Setting up Router services</Description>
<Priority>3-Medium</Priority>
<SRNumber>1-MYUNIQUEVALUE</SRNumber>
<ServiceRequestType>External</ServiceRequestType>
<ListOfAction>
<Action>
<BillableFlag>N</BillableFlag>
<Description2>test activity1</Description2>
<EstWorkTimeRemaining>8</EstWorkTimeRemaining>
<Planned/>
<PrimaryOwnedBy>SADMIN</PrimaryOwnedBy>
<RowStatusOld>N</RowStatusOld>
>Status>Unscheduled</Status>
>Type>Appointment</Type>
</Action>
<Action>
<BillableFlag>N</BillableFlag>
<Description2>test activity2</Description2>
<EstWorkTimeRemaining>8</EstWorkTimeRemaining>
<Planned/>
<PrimaryOwnedBy>SADMIN</PrimaryOwnedBy>
<Status>Unscheduled</Status>
<Type>Appointment</Type>
</Action>
</ListOfAction>
</ServiceRequest>
</ListOfSampleServiceRequest>
</SiebelMessage>

For this example to work, you must have the contact, Kastrup-Larsen, in the database. If you try the Insert method against a server database where the contact does not exist, then you will receive the following error:

Picklist validation of field 'Contact Last Name' in integration component 'Service Request' did not find any matches satisfying the query '[Last Name] = "Kastrup-Larsen"', and an attempt to create a new record through the picklist failed (SBL-EAI-04186)

Also, if you try to insert the previous instance a second time, then you will receive the following error message:

158 Integration Platform Technologies: Siebel Enterprise Application Integration Version 8.1/8.2
IDS_ERR_EAI_SA_INSERT_MATCH_FOUND. Insert operation on integration component 'Service Request' failed because a matching record in business component 'Service Request' with search specification '[SR Number] = '1-MYUNIQUEVALUE' was found. (SBL-EAI-04383).

Upsert Method

The Upsert method is similar to the Synchronize method with one exception; the Upsert method does not delete any records.

The Upsert method will result in insert or update operations. If the record exists, then it will be updated. If the record does not exist, then it will be inserted. Unlike the Synchronize method, upsert will not delete any children.

To determine if an update or insert is performed, the EAI Siebel Adapter runs a query using user keys fields or the search specifications to determine if the parent or primary record already exists. If the parent record exists, it will be updated. If no matching parent record is found, then the new record will be inserted. Once again, upsert will not delete any children. If existing children are found, then they are updated.

You can update multiple corresponding top-level parent business component records using one XML file, as in the following example:

```xml
<SiebelMessage MessageId="" MessageType="Integration Object" IntObjectName="Transaction">
  <ListofTransaction>
    <Transaction>
      <Field1>xxx</Field1>
      <Field2>yyy</Field2>
      ...
    </Transaction>
    <Transaction>
      <Field1>aaa</Field1>
      <Field2>bbb</Field2>
      ...
    </Transaction>
  </ListofTransaction>
</SiebelMessage>
```

Update Method

This method is similar to the Synchronize method, except that the EAI Siebel Adapter returns an error if no match is found for the root component; otherwise, it updates the matching record and synchronizes all the children.

**NOTE:** During an update operation, the EAI Siebel Adapter expects a single record to be returned from the user key search. If more than one record is returned, then EAI Siebel Adapter throws an error.
For example, if you send an order with one order item to the EAI Siebel Adapter, then it will take the following actions:

1. Queries for the order, and if it finds a match, then it updates the record.
2. Updates or inserts the new order item depending on whether a match was found for the new order item.
3. Deletes any other order items associated with that order.

## Delete Method

You can delete one or more records in a business component that is mapped to the root integration component, given an integration object instance. A business component record is deleted as specified by an integration object instance. The integration component instance fields are used to query the corresponding business component and any records retrieved will be deleted. You call the Delete method using only one of the following method arguments:

- A Query By Example (QBE) integration object instance.
- A Primary Row Id and Output Integration Object Name.
- A Search Specification.

**NOTE:** To have the EAI Siebel Adapter perform a delete operation, define an integration object that contains the minimum fields on the primary business component for the business object. The EAI Siebel Adapter attempts to delete matching records in the business component before deleting the parent record.

For example, if you pass in this XML document, then the Test Account account is deleted.

```xml
<?xml version = "1.0" encoding = "UTF-8"?>
<Siebel-Property-Set EscapeNames="false">
<SiebelMessage MessageId = "1-2IOY" IntObjectName = "EAI Account" MessageType = "Integration Object" IntObjectFormat = "Siebel Hierarchical">
<ListOfAccount>
  <Account>
    <Name>Test Account</Name>
    <Location>EMV</Location>
  </Account>
</ListOfAccount>
</SiebelMessage>
```

Any child contacts that once belonged to the account will still remain in the database, but will not be associated with this Account.

## Execute Method

The Execute method can be specified on the EAI Siebel Adapter to perform combinations of various operations on components in an integration object instance. This method uses the following operations:
NOTE: A none operation is equivalent to operation sync.

These operations perform the same tasks as the related methods. For example, the delete operation makes the EAI Siebel Adapter delete the business component record matched to the particular integration component instance. However, what will be done to the children depends on the combination of the parent operation and the child operation. For information, see Table 16 on page 162.

NOTE: The operation method names are case sensitive. If you misspell an operation method, then the EAI Siebel Adapter assumes the default operation.

An XML document sent to a Siebel application can include operations that describe whether a particular data element must be inserted, updated, deleted, synchronized, and so on. These operations can be specified as an attribute at the component level. They cannot be specified for any other element.

The following XML example demonstrates using the upsert and delete operation to delete a particular child record without updating the parent:

```xml
<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">
          ...
        </Contact>
      </ListofContact>
    </Account>
  </ListofSampleAccount>
</SiebelMessage>
```
<First Name>Stan</First Name>
<JobTitle>Senior Mgr of MIS</JobTitle>
<LastName>Graner</LastName>
<MiddleName>A</MiddleName>
<Personal Contact>N</Personal Contact>
<Account>A. K. Parker Distribution</Account>
<Account Location>HQ-Distribution</Account Location>
</Contact>
</ListOfContact>
</Account>
</ListOfSampleAccount>
</SiebelMessage>

About Execute Method Operations
Specify an attribute named operation, in lowercase, to the component’s XML element. The legal values for this attribute are upsert, sync, delete, query, update, insert, updatesync, insertsync, skipnode, skiptree, and none. If the operation is not specified on the root component, then the sync operation is used as the default.

**NOTE:** Specifying an operation within the ListOf tag is not supported. For information on the ListOf tag, see XML Reference: Siebel Enterprise Application Integration.

Each child node inherits the operation from the parent if another operation is not explicitly specified. If another operation is explicitly specified, then Table 16 represents the results of the operation on the current node.

**Table 16. Operation Outcomes for the Child Node**

<table>
<thead>
<tr>
<th>Operation</th>
<th>What Happens to the Current Node</th>
<th>What Happens to Unmatched Children of Current Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>upsert</td>
<td>Update or insert</td>
<td>Leave alone</td>
</tr>
<tr>
<td>sync</td>
<td>Update or insert</td>
<td>Delete</td>
</tr>
<tr>
<td>update</td>
<td>Update</td>
<td>Delete</td>
</tr>
<tr>
<td>updatesync</td>
<td>Update</td>
<td>Delete</td>
</tr>
<tr>
<td>insert</td>
<td>Insert</td>
<td>Leave alone</td>
</tr>
<tr>
<td>insertsync</td>
<td>Insert</td>
<td>Delete</td>
</tr>
<tr>
<td>skipnode</td>
<td>Skip this node</td>
<td>Leave alone</td>
</tr>
<tr>
<td>skiptree</td>
<td>Skip the tree</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
Example of a Parent Node Using a Sync Operation

This example demonstrates the effects of records after a sync operation is performed. Table 17 is a high level representation of a parent node using the sync operation of the Execute method.

Table 17. Representation of a Parent Node Using the Sync Operation

<table>
<thead>
<tr>
<th>Record In Database</th>
<th>Integration Object Instance</th>
<th>Record After Execute Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account1</td>
<td>Account1 operation=sync</td>
<td>Account1</td>
</tr>
<tr>
<td>Contact0</td>
<td>Contact1</td>
<td>Contact1</td>
</tr>
<tr>
<td>Contact1</td>
<td>Contact2</td>
<td></td>
</tr>
</tbody>
</table>

Record in Database

The following code represents GENCOMM0 and GENCOMM1 being retrieved as the contacts for this example:

```xml
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType = "Integration Object" IntObjectFormat = "Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <AccountStatus>Active</AccountStatus>
      <CurrencyCode>USD</CurrencyCode>
      <LanguageCode>ENU</LanguageCode>
      <Location>San Francisco, CA</Location>
      <Name>GenComm</Name>
      <ListOfContact>
        <Contact>
          <FirstName>GENCOMM0</FirstName>
          <LastName>GENCOMM0</LastName>
          <MiddleName/>
          <Organization>Default Organization</Organization>
        </Contact>
        <Contact>
          <FirstName>GENCOMM1</FirstName>
          <LastName>GENCOMM1</LastName>
          <MiddleName/>
          <Organization>Default Organization</Organization>
        </Contact>
      </ListOfContact>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```

Integration Object Instance

The following code represents the sync operation acting on the contacts from the database.
EAI Siebel Adapter Business Service ■ EAI Siebel Adapter Business Service Methods

<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType = "Integration Object" IntObjectFormat = "Siebel Hierarchical">
<ListOfAccount>
<Account operation="sync">
<AccountStatus>Inactive</AccountStatus>
<CurrencyCode>USD</CurrencyCode>
<LanguageCode>ENU</LanguageCode>
<Location>San Francisco, CA</Location>
<Account>
<Contact>
<FirstName>GENCOMM1</FirstName>
<LastName>GENCOMM1</LastName>
<Organization>Default Organization</Organization>
</Contact>
<Contact>
<FirstName>GENCOMM2</FirstName>
<LastName>GENCOMM2</LastName>
<Organization>Default Organization</Organization>
</Contact>
</Account>
</ListOfAccount>
</SiebelMessage>

Result Record in Database
The following code represents the results of the sync operation after acting on the two contacts from the database.

<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType = "Integration Object" IntObjectFormat = "Siebel Hierarchical">
<ListOfAccount>
<Account>
<AccountStatus>Inactive</AccountStatus>
<CurrencyCode>USD</CurrencyCode>
<LanguageCode>ENU</LanguageCode>
<Location>San Francisco, CA</Location>
<Account>
<Contact>
<FirstName>GENCOMM1</FirstName>
<LastName>GENCOMM1</LastName>
<Organization>Default Organization</Organization>
</Contact>
<Contact>
<FirstName>GENCOMM2</FirstName>
<LastName>GENCOMM2</LastName>
<Organization>Default Organization</Organization>
</Contact>
</Account>
</ListOfAccount>
</SiebelMessage>
In this case, if a matching Account1 exists in the database, then the EAI Siebel Adapter will perform an update of that record. If no record matching Account1 exists, then the EAI Siebel Adapter will insert a new account.

For all the matching child contacts, the sync operation is inherited. Therefore, if the child exists, then it will be updated. If the child does not exist, then it is inserted. Any child contacts that exist in the database but do not match the integration object instance (unmatched children) are deleted.

The reason for this logic is that the sync operation makes the record in the database look like the integration object instance.

**Example of a Parent Node Using an Update Operation**

This example demonstrates the effects of records after an update operation is performed. Table 18 is a high level representation of a parent node using the update operation of the Execute method.

**NOTE:** The examples represented by Table 18, Table 19 on page 166, and Table 22 on page 167 basically have the same result. However, as reflected in Table 21 on page 167, the children do not automatically inherit *Update* if it is only set for the root.

Table 18. Representation of a Parent Node Using the Update Method

<table>
<thead>
<tr>
<th>Record In Database</th>
<th>Integration Object Instance</th>
<th>Record After Execute Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account1</td>
<td>Account1 operation=update</td>
<td>Account1</td>
</tr>
<tr>
<td>Contact0</td>
<td>Contact1</td>
<td>Contact1</td>
</tr>
<tr>
<td>Contact1</td>
<td>Contact2</td>
<td>Contact2</td>
</tr>
</tbody>
</table>

In this case, if a record matching Account1 exists in the database, then the EAI Siebel Adapter updates that specific record. If no matching account exists, then the result of the EAI Siebel Adapter is an error with this message:

`Insert operation on integration component 'Account' failed because a matching record in business component 'Account' with search specification '[Name] = "GenComm" AND [Location] = 'San Francisco, CA'' was found (SBL-EAI-04383)`

For all the matching child contacts, the update operation is inherited. Therefore, if the child exists, then it will be updated. If the child does not exist, then it is inserted. For child contacts that exist in the database but do not match the integration object instance, they will be deleted. These might be child contacts created or associated with the Account by default.
This is very similar to the previous example except the record must exist in the database.

**Example of a Parent Using an Update Operation and One More Child Using an Insert Operation**

This example demonstrates the effects on records after an update operation acts on the parent, and an insert operation acts on one of the children records. Table 19 is a high level representation of this example.

Table 19. Representation of Two Operations

<table>
<thead>
<tr>
<th>Record In Database</th>
<th>Integration Object Instance</th>
<th>Record After Execute Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account1</td>
<td>Account1 operation=update</td>
<td>Account1</td>
</tr>
<tr>
<td></td>
<td>Contact1</td>
<td>Contact1</td>
</tr>
<tr>
<td></td>
<td>Contact2 operation=insert</td>
<td></td>
</tr>
</tbody>
</table>

In this case, if a record matching Account1 exists in the database, then the EAI Siebel Adapter updates that record. If no record matching Account1 exists, then the result from the EAI Siebel Adapter is an error.

You can also override the parent operation as in the case for Contact2. Since Contact2 does not exist, and there is an explicit insert operation, it will be inserted. Any unmatched children will be deleted as part of the parent operation (update). This is the reason why Contact0 is deleted.

If you are explicitly overriding the parent operation, then you must make sure the operation applies. For example, the two combinations in Table 20 and Table 21 on page 167 will fail. In Table 20, it fails because an insert is attempted when Contact1 already exists in the database.

Table 20. Representation of Overriding a Parent Operation Using Insert

<table>
<thead>
<tr>
<th>Record In Database</th>
<th>Integration Object Instance</th>
<th>Record After Execute Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account1</td>
<td>Account1 operation=update</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Contact1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Contact2 operation=insert</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Contact2 operation=insert</td>
<td></td>
</tr>
</tbody>
</table>
In Table 21, the update fails because SubContact3 inherits from Contact2's operation, and Subcontact3 does not exist in the database.

Table 21. Representation of Overriding a Parent Operation Using Update

<table>
<thead>
<tr>
<th>Record In Database</th>
<th>Integration Object Instance</th>
<th>Record After Execute Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account1</td>
<td>Account1</td>
<td>None</td>
</tr>
<tr>
<td>Contact1</td>
<td>Contact1</td>
<td></td>
</tr>
<tr>
<td>Contact2</td>
<td>Contact2 operation=update</td>
<td></td>
</tr>
<tr>
<td>SubContact1</td>
<td>SubContact1</td>
<td></td>
</tr>
<tr>
<td>SubContact2</td>
<td>SubContact3</td>
<td></td>
</tr>
</tbody>
</table>

Example of a Parent Using the Update Operation and One More Child Using the Upsert Operation

This example demonstrates the effects of records after an update operation acts on the parent, and an upsert operation acts on one of the children records. Table 22 is a high level representation of this example.

Table 22. Representation of Overriding a Parent Operation Using Upsert

<table>
<thead>
<tr>
<th>Record In Database</th>
<th>Integration Object Instance</th>
<th>Record After Execute Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account1</td>
<td>Account1 operation=update</td>
<td>Account1</td>
</tr>
<tr>
<td>Contact0</td>
<td>Contact1</td>
<td>Contact1</td>
</tr>
<tr>
<td>Contact1</td>
<td>Contact2 operation=upsert</td>
<td>Contact2</td>
</tr>
</tbody>
</table>

In this case, if a record matching Account1 exists in the database, then the EAI Siebel Adapter updates that record. If no record matching Account1 exists, then the result of the EAI Siebel Adapter is an error.

For a record matching Contact2, the upsert operation overrides the update operation. Therefore, if Contact2 exists, then it is updated. If no record matching Contact2 is found, then it is inserted. Unmatched child contacts are deleted.
Example of a Parent Using the Upsert Operation and One More Child Using the Sync Operation

This example demonstrates the effects of records after an update operation acts on the parent, and a sync operation acts on one of the children records. Table 23 is a high level representation of this example.

Table 23. Representation of Overriding a Parent Operation Using Sync

<table>
<thead>
<tr>
<th>Record In Database</th>
<th>Integration Object Instance</th>
<th>Record After Execute Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account1</td>
<td>Account1 operation=upsert</td>
<td>Account1</td>
</tr>
<tr>
<td>Contact0</td>
<td>Contact1</td>
<td>Contact0</td>
</tr>
<tr>
<td>Organization2</td>
<td>Organization1</td>
<td>Organization2</td>
</tr>
<tr>
<td>Contact1</td>
<td>Contact2 operation=sync</td>
<td>Contact1</td>
</tr>
<tr>
<td>Organization2</td>
<td>Organization3</td>
<td>Organization1</td>
</tr>
<tr>
<td>Contact2</td>
<td>Organization2</td>
<td>Organization2</td>
</tr>
<tr>
<td>Organization2</td>
<td></td>
<td>Organization3</td>
</tr>
</tbody>
</table>

In this case, if a record matching Account1 exists in the database, then the EAI Siebel Adapter updates that record. If no record matching Account1 exists, then the EAI Siebel Adapter inserts the record.

For all child contacts, the upsert operation applies. Therefore, if the child exists, then it is updated. If the child does not exist, then it is inserted. For child contacts that exist in the database, but do not match the integration object instance, they will remain unchanged because upsert does not delete children.

In the case of Contact2, which has the sync operation overriding the upsert operation, it is updated, and its children are synchronized.

Skiptree Operation

The whole sub tree rooted at this node is not processed. It is the same as that whole sub tree not existing in the integration object instance. Operations specified in child nodes do not affect processing in any way since the EAI Siebel Adapter does not act on the child.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId="1-2RE" MessageType="Integration Object" IntObjectName="Sample Account" IntObjectFormat="Siebel Hierarchical">
  <ListOfSampleAccount>
    <Account operation="upsert">
      <Name>foo</Name>
      <Location>cold storage</Location>
    </Account>
  </ListOfSampleAccount>
</SiebelMessage>
```
Based on this example, the account is upserted. The processing of the first contact is completely skipped although the business address child has an `insert` operation set. Also, the second contact is upserted.

If the skiptree operation is specified for the account integration component, then the EAI Siebel Adapter skips processing the complete account. This results in no operation. It is possible to have many accounts with some having skiptree specified as shown in the following example. The EAI Siebel Adapter processes the trees that do not have skiptree specified.

```xml
<?xml version='1.0' encoding='UTF-8'?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId="1-2RE" MessageType="Integration Object" IntObjectName="Sample Account" IntObjectFormat="Siebel Hierarchical">
<ListOfSampleAccount>
  <Account operation="skiptree">
    <Name>foo</Name>
    <Location>cold storage</Location/>
  </Account>
  <Account operation="upsert">
    <Name>bar</Name>
    <Location>cold storage</Location/>
  </Account>
</ListOfSampleAccount>
</SiebelMessage>
```
Skipnode Operation

Similar to all other Execute operations, the children nodes inherit the semantics of the operation from the parent nodes. If a node has the skipnode operation set, then the EAI Siebel Adapter will skip setting field values for all children unless a child has an explicit operation set that will override.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId="1-2RE" MessageType="Integration Object"
IntObjectName="EAI Account" IntObjectFormat="Siebel Hierarchical">
<ListOfAccount>
  <Account operation="skipnode">
    <Name>foo</Name>
    <Location>cold storage</Location>
    <ListOfContact>
      <Contact operation="upsert">
        <IntegrationId>1-123</IntegrationId>
        <FirstName>firstname</FirstName>
        <LastName>contact1</LastName>
        <ListOfContact_Organization>
          <Contact_Organization>
            <Organization operation="insert">MyOrganization</Organization>
          </Contact_Organization>
        </ListOfContact_Organization>
      </Contact>
      <Contact operation="upsert">
        <IntegrationId>2-123</IntegrationId>
        <FirstName>firstname</FirstName>
        <LastName>contact2</LastName>
      </Contact>
    </ListOfContact>
  </Account>
</ListOfAccount>
</SiebelMessage>
```

Based on this example, the account is skipped. However, the EAI Siebel Adapter will attempt to insert the two contacts.
EAI Siebel Adapter Business Service Method Arguments

Each of the EAI Siebel Adapter methods takes arguments that allow you to specify required and optional information to the adapter. You can locate the arguments for each method (and whether it can be used as an input argument, output argument, or both) in Table 24.

Table 24. EAI Siebel Adapter Business Service Method Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Query</th>
<th>Query Page</th>
<th>Sync</th>
<th>Upsert</th>
<th>Update</th>
<th>Insert</th>
<th>Delete</th>
<th>Execute</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusObjCacheSize</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
</tr>
<tr>
<td>DeleteByUserKey</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Input</td>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>ErrorOnNonExisting</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Input</td>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>Delete</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Input</td>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>ExecutionMode</td>
<td>Input</td>
<td>Input</td>
<td>Not applicable</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Not applicable</td>
<td>Input</td>
</tr>
<tr>
<td>IntObjectName</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Input</td>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>LastPage</td>
<td>Not applicable</td>
<td>Output</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Output</td>
</tr>
<tr>
<td>MessageId</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
</tr>
<tr>
<td>NewQuery</td>
<td>Not applicable</td>
<td>Input</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Input</td>
</tr>
<tr>
<td>NumOutputObjects</td>
<td>Output</td>
<td>Output</td>
<td>Output</td>
<td>Output</td>
<td>Output</td>
<td>Output</td>
<td>Output</td>
<td>Output</td>
</tr>
<tr>
<td>OutputIntObjectName</td>
<td>Input</td>
<td>Input</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Input</td>
</tr>
<tr>
<td>PageSize</td>
<td>Not applicable</td>
<td>Input</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Input</td>
</tr>
</tbody>
</table>
Table 24. EAI Siebel Adapter Business Service Method Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Query</th>
<th>Query Page</th>
<th>Sync</th>
<th>Upsert</th>
<th>Update</th>
<th>Insert</th>
<th>Delete</th>
<th>Execute</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrimaryRowId</td>
<td>Input</td>
<td>Not applicable</td>
<td>Output</td>
<td>Output</td>
<td>Output</td>
<td>Output</td>
<td>Input</td>
<td>Input/Output</td>
</tr>
<tr>
<td>QueryByUserKey</td>
<td>Input</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Input</td>
</tr>
<tr>
<td>SearchSpec</td>
<td>Input</td>
<td>Input</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Input</td>
</tr>
<tr>
<td>SiebelMessage</td>
<td>Input/Output</td>
<td>Output</td>
<td>Input/Output</td>
<td>Input/Output</td>
<td>Input/Output</td>
<td>Input/Output</td>
<td>Input/Output</td>
<td></td>
</tr>
<tr>
<td>SortSpec</td>
<td>Not applicable</td>
<td>Input</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Input</td>
</tr>
<tr>
<td>StartRowNum</td>
<td>Not applicable</td>
<td>Input</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Input</td>
</tr>
<tr>
<td>StatusObject</td>
<td>Not applicable</td>
<td>Not applicable</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td></td>
</tr>
<tr>
<td>ViewMode</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td>Input</td>
<td></td>
</tr>
</tbody>
</table>

Table 25 presents each argument of the EAI Siebel Adapter business service methods.

Table 25. Defining EAI Siebel Adapter Business Service Method Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Display Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusObjCacheSize</td>
<td>Business Object Cache Size</td>
<td>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.</td>
</tr>
<tr>
<td>DeleteByUserKey</td>
<td>Delete By User Key</td>
<td>A Boolean argument. Forces the EAI Siebel Adapter to use only the user keys to identify a record.</td>
</tr>
<tr>
<td>ErrorOnNonExistingDelete</td>
<td>Error On Non Existing Delete</td>
<td>A Boolean argument. Determines whether or not the EAI Siebel Adapter aborts the operation if no match is found.</td>
</tr>
</tbody>
</table>
Table 25. Defining EAI Siebel Adapter Business Service Method Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Display Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExecutionMode</td>
<td>Execution Mode</td>
<td>Used to set the direction of a query on a business component. Valid values are ForwardOnly and Bidirectional. The default is Bidirectional. ForwardOnly is more efficient than Bidirectional, and is recommended in cases where you must process a large number of records in the forward direction only (such as for report generation). For operations that are likely to return more than 10000 records, use ForwardOnly to avoid errors. For more information on executing queries, see the topic on the ExecuteQuery business component method in Siebel Object Interfaces Reference.</td>
</tr>
<tr>
<td>IntObjectName</td>
<td>Integration Object Name</td>
<td>Name of the integration object to delete.</td>
</tr>
<tr>
<td>LastPage</td>
<td>Last Page</td>
<td>Boolean indicating whether or not the last record in the query result set has been returned.</td>
</tr>
<tr>
<td>MessageId</td>
<td>Message Id</td>
<td>The MessageId can be used to specify the ID for the generated message. By default, the EAI Siebel Adapter generates a unique ID for each message. However, if you want to use the workflow instance ID, then you can use this argument to specify the ID.</td>
</tr>
<tr>
<td>NewQuery</td>
<td>New Query</td>
<td>Default is False. Boolean indicating whether a new query will be executed. If set to True, a new query is executed flushing the cache for that particular integration object.</td>
</tr>
<tr>
<td>NumOutputObjects</td>
<td>Number of Output Integration Objects</td>
<td>Number of output integration objects.</td>
</tr>
<tr>
<td>OutputIntObjectName</td>
<td>Output Integration Object Name</td>
<td>The name of the integration object that is to be output.</td>
</tr>
<tr>
<td>PageSize</td>
<td>Page Size</td>
<td>Default is 10. Indicates the maximum number of integration object instances to be returned.</td>
</tr>
</tbody>
</table>
### EAI Siebel Adapter Business Service Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Display Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrimaryRowId</td>
<td>Object Id</td>
<td>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.</td>
</tr>
<tr>
<td>QueryByUserKey</td>
<td>Query By Key</td>
<td>A Boolean argument. Forces the EAI Siebel Adapter to use only the user keys to perform a query.</td>
</tr>
<tr>
<td>SearchSpec</td>
<td>Search Specification</td>
<td>This argument allows you to specify complex search specifications as free text in a single method argument. For information, see &quot;About Using Language-Independent Code with the EAI Siebel Adapter Business Service&quot; on page 185.</td>
</tr>
<tr>
<td>SiebelMessage</td>
<td>Siebel Message</td>
<td>The input or the output integration object instance.</td>
</tr>
<tr>
<td>SortSpec</td>
<td>Sort Specification</td>
<td>Default is the SortSpec of the underlying business component. This argument allows you to specify complex sort criteria as a free text in a single method argument, using any business component fields and standard Siebel sort syntax. For examples, see Using Siebel Tools.</td>
</tr>
<tr>
<td>StartRowNum</td>
<td>Starting Row Number</td>
<td>Default is 0 (first page). Indicates the row in the result set for the QueryPage method to start retrieving a page of records.</td>
</tr>
<tr>
<td>StatusObject</td>
<td>Status Object</td>
<td>This argument tells the EAI Siebel Adapter whether or not to return a status message.</td>
</tr>
<tr>
<td>ViewMode</td>
<td>View Mode</td>
<td>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.</td>
</tr>
</tbody>
</table>
**About the SearchSpec Input Method Argument**

The SearchSpec input method argument is applicable to the QueryPage, Query, Delete, and Execute methods. This method argument allows you to specify complex search specifications as free text in a single method argument. Expressions within a single integration component are restricted only by the Siebel Query Language supported by the Object Manager. Integration components and fields are referenced using the following notation:

```
[IntCompName.Int CompField Name]
```

For example, given an integration object definition with two integration components, Account as the root component and Contact as the child component, the following search specification is allowed:

```
([Account.Site] LIKE "A*" OR [Account.Site] IS NULL) AND [Contact.PhoneNumber] IS NOT NULL
```

This search specification queries accounts that either have a site that starts with the character A, or do not have a site specified. In addition, for the queried accounts, it queries only those associated contacts who have a phone number.

**NOTE:** The operator between fields for a particular integration component instance can be AND unless between the same field. You use the DOT notation to refer to integration components and their fields.

You can include the child integration component in a search specification only if its parent components are also included.

**About Multivalue Groups in the EAI Siebel Adapter Business Service**

Multivalue groups (MVGs) in the business components are mapped to separate integration components. Such integration components are denoted by setting a user property MVG on the integration component to Y. For information on MVGs, see Chapter 2, “Integration Objects.”

An integration component instance that corresponds to a primary MVG is denoted by the attribute IsPrimaryMVG set to Y. This attribute is a hidden integration component field and does not have a corresponding business component field.

Each MVG that appears on the client UI is mapped to a separate integration component. For example, in the Orders Entry - Orders screen, there is an account address, a bill-to address, and a ship-to address. Each of these MVGs needs a separate integration component definition. Each field defined for an integration component (represented by the class CSSEAIIntCompFieldDef) maps to a field in the MVG. For such fields, External Name denotes the name of the business component field as it appears on the master business component, and the user property MVGFieldName denotes the name of the business component field as it appears on the MVG business component.

**NOTE:** Setting a primary record in an MVG is supported when the Auto Primary property of the underlying multivalue link is specified as Selected, None, or Default.
Setting a Primary Position for a Contact
You have a contact with multiple contact positions in a Siebel application. None of these positions are marked as the primary position for the contact, and you want to select one of them as the primary position.

To specify a contact position as a primary
1. Create your XML file and insert `<IsPrimaryMVG='Y'>` before the contact position you want to identify as the primary position for the contact as follows:

   ```xml
   <?xml version="1.0" encoding="UTF-8" ?>
   <Siebel-Property-Set EscapeNames="false">
   - <SiebelMessage MessageId="1-69A" IntObjectName="Sample Contact">
     - <ListOfSampleContact>
       - <Contact>
         - <FirstName>Pal888</FirstName>
         - <IntegrationId>65454398</IntegrationId>
         - <JobTitle>Manager</JobTitle>
         - <LastName>John888</LastName>
         - <MiddleName/>
         - <PersonUId>1-Y88H</PersonUId>
         - <PersonalContact>N</PersonalContact>
       - <ListOfContact_Position>
         - <Contact_Position IsPrimaryMVG="Y">
           - <EmployeeFirstName>Siebel</EmployeeFirstName>
           - <EmployeeLastName>Administrator</EmployeeLastName>
           - <Position>Siebel Administrator</Position>
           - <RowStatus>N</RowStatus>
           - <SalesRep>SADMIN</SalesRep>
         - </Contact_Position>
       - </ListOfContact_Position>
     - </Contact>
   - </ListOfSampleContact>
   </SiebelMessage>.
   ```

2. Use the Upsert or Sync method to update the account.

About Using Effective Dating with Siebel EAI Adapter Business Service
The Siebel EAI Adapter allows you to access effective dating data, which means the start date and end date for a given field or link. Third-party applications can request and receive effective dating data from the Siebel application.

To view XML samples for effective dating functionality see XML Reference: Siebel Enterprise Application Integration.
You specify effective dating on fields of a given business component through a Siebel Web Client administration screen. For more information, see "Enabling Effective Dating on Fields" on page 177. Two integration component field attributes allow you to set effective dating: EDStartDate and EDEndDate. Standard querying techniques, such as query, insert, update, and sync can be used to request effective dating-enabled data.

As Figure 33 shows, the Siebel Object Manager framework features APIs that are called by the Siebel EAI Adapter when an integration object with effective dating-enabled fields or links is read by the XML Converter.

Figure 33. Effective Dating Overview

The following two topics explain how effective dating works:

- "Enabling Effective Dating on Fields" on page 177
- "Enabling Effective Dating on Links" on page 181

**Enabling Effective Dating on Fields**

This topic explains how effective dating works on fields. It contains the following topics:

- "Configuring Integration Components for Effective Dating on Fields" on page 177
- "How the XML Converter Reads Effective Dating Data from Fields" on page 179
- "About WSDL Schema Generation for Effective Dating on Fields" on page 180

**Configuring Integration Components for Effective Dating on Fields**

You can enable effective dating on a field in an existing integration object if the corresponding field in the business component is effective dating-enabled.
Enabling Effective Dating on Fields
You enable effective dating on fields in the Administration - Effective Dating screen.

To enable effective dating on fields
1. In the Siebel Web Client, navigate to Administration - Effective Dating, then Field.
2. In the Effective Dating Buscomp list, select the required business component.
   - The list displays the fields already enabled for effective dating.
3. If you want to enable effective dating field not present in this list, click New, then in the Field field click the Select button.
4. In the Business Component Fields window select the required fields, then click OK.
   - This enables the fields for effective dating.

Enabling Effective Dating on Integration Objects
You use Siebel Tools to synchronize the object from your current repository, with its underlying business object in the Siebel database which contains the new EDEnabled user property. For more information on synchronizing integration objects, see "Synchronizing Integration Objects" on page 61.

To enable effective dating on an integration object
1. In Siebel Tools, select the integration object you want to enable for effective dating.
2. Click the Synchronize button in the Integration Objects list.
   - The Integration Object Synchronize wizard appears.
3. Click the plus symbol to display all the related integration components.
4. Uncheck the boxes beside the objects and components you do not want to include in the synchronization of your Siebel integration object.
5. Locate the integration component containing the effective dating-enabled fields and drill down on it.
6. Click the plus symbol to display all the user properties of the field.
7. Locate the EDEnabled user property from the list, select it, then and add it to the repository side.
8. Review the summary, and if changes are needed, click Back and make the needed changes.
9. If no changes are needed, click Finish to synchronize the Siebel integration object and the Siebel business object.
How the XML Converter Reads Effective Dating Data from Fields

The XML converter reads effective dating attributes contained in an XML file, maps them to a property set, and converts the property set to an integration object instance by embedding the EDStartDate and EDEndDate attributes inside the field name.

For example: If a SOAP request contains the following query for an effective dating field:

```xml
<acc:EDListOfFirstName>
  <acc:FirstName EDStartDate="04/01/2012" EDEndDate="">James</acc:FirstName>
</acc:EDListOfFirstName>
```

The data will be converted into a child property set of the integration component instance as follows:

```xml
{[0] CCFPropertySet@1DA79960 p#0 c#1 type="ListOfRelated Contact" vt=0 value="
  {[0] CCFPropertySet@1D9FD870 p#1 c#2 type="Related Contact" vt=0 value="
    {"Contact Id"} = "Contact1";
    {[0] CCFPropertySet@1FD92B80 p#0 c#1 type="EDListOfFirstName" vt=0 value="
      {[0] CCFPropertySet@13258470 p#2 c#0 type="FirstName" vt=3 value="James"
        {["EDEndDate"] = "";
        {["EDStartDate"] = "04/25/2012";}
      }
    }
  }
}
```

The dates are now embedded into the field name.
About WSDL Schema Generation for Effective Dating on Fields

Effective dating requires two complex type attributes StartDate and EndDate for each effective dating-enabled field. In the following schema example the location field is enabled for effective dating, as shown by the two additional attributes EDStartDate and EDEndDate. Historical data can be retrieved by setting the cardinality of the effective dating-enabled XSD element to unbounded.

WSDL Schema Example

```xml
<xsd:complexType name="RelatedContact">
  <xsd:sequence>
    <xsd:element name="ContactId" maxOccurs="1" minOccurs="0" type="xsd:string"/>
    <xsd:element name="EDListOfFirstName" maxOccurs="1" minOccurs="0" type="xsdLocal1:EDListOfFirstName"/>
    <xsd:element name="ContactIntegrationId" maxOccurs="1" minOccurs="0" type="xsd:string"/>
    <xsd:element name="EDListOfLastName" maxOccurs="1" minOccurs="0" type="xsdLocal1:EDListOfLastName"/>
  </xsd:sequence>
</xsd:complexType>
```

```xml
<xsd:complexType name="EDListOfFirstName">
  <xsd:sequence>
    <xsd:element name="FirstName" maxOccurs="unbounded" minOccurs="0" type="xsdLocal1:FirstName"/>
  </xsd:sequence>
</xsd:complexType>
```

```xml
<xsd:complexType name="FirstName">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="EDStartDate" type="xsd:string"/>
      <xsd:attribute name="EDEndDate" type="xsd:string"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```
SOAP Query Example
The previous WSDL schema example allows you to generate the following SOAP query:

```xml
<Element Name EDStartDate=dd1/mm1/yyyy1, EDEndDate = dd2/mm2/yyyy2>value</Element Name>
```

For example:

```xml
<acc:RelatedContact>
  <acc:ContactId>88-30ARL</acc:ContactId>
  <acc:EDListOfFirstName>
    <acc:FirstName EDStartDate="04/01/2012" EDEndDate="">James</acc:FirstName>
  </acc:EDListOfFirstName>
  <acc:EDListOfLastName>
    <acc:LastName EDStartDate="04/01/2012" EDEndDate="">Bond</acc:LastName>
  </acc:EDListOfLastName>
</acc:RelatedContact>
```

Similarly you can perform insert, update and synchronize operations on data using the "WSDL Schema Example" on page 180.

Enabling Effective Dating on Links
This topic explains how to enable effective dating on links. It contains the following topics:

- "Enabling Effective Dating on Links" on page 182
- "About Siebel EAI Adapter Operations for Effective Dating on Links" on page 184
Enabling Effective Dating on Links

You can enable effective dating on a link in an existing integration object if the corresponding link in the business component is effective dating-enabled. You enable link for effective dating through the Siebel Web Client administration Effective Dating screen.

To enable effective dating on links

1. In the Siebel Web Client, navigate to Administration - Effective Dating, then Child Buscomp.
2. In the Effective Dating Buscomp list, select the required business component, then in the Child Buscomp view select the required link if it is shown.
   If you need to create a new link, see Step 3.
3. Click New, then in the Link Name field click the Select button.
4. In the Link window select the required link, then click OK.

Web Service Schema Example

The following Web service schema example shows a link between the Household and Related Contact business components which have been enabled for effective dating. The effective dating attributes are displayed in bold text.

```xml
<xsd:complexType name="Household">
  <xsd:complexType name="RelatedContact">
    <xsd:attribute name="EDStartDate" type="xsd:string" />
    <xsd:attribute name="EDEndDate" type="xsd:string" />
    <xsd:sequence>
      <xsd:element name="ContactIntegrationId" maxOccurs="1" minOccurs="0" type="xsd:string" />
      <xsd:element name="EDListOfFirstName" maxOccurs="1" minOccurs="0" type="xsdLocal1:EDListOfFirstName" />
      <xsd:element name="EDListOfLastName" maxOccurs="1" minOccurs="0" type="xsdLocal1:EDListOfLastName" />
      <xsd:element name="MiddleName" maxOccurs="1" minOccurs="0" type="xsd:string" />
      <xsd:element name="PersonalContact" maxOccurs="1" minOccurs="0" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:complexType>
```
<xsd:element name="ContactId" maxOccurs="1" minOccurs="0" type="xsd:string" />
<xsd:element name="DateEnteredHousehold" maxOccurs="1" minOccurs="0" type="xsd:string" />
<xsd:element name="DateExitedHousehold" maxOccurs="1" minOccurs="0" type="xsd:string" />
<xsd:element name="PrimaryOrganizationId" maxOccurs="1" minOccurs="0" type="xsd:string" />
<xsd:element name="Relationship" maxOccurs="1" minOccurs="0" type="xsd:string" />
</xsd:sequence>
<xsd:attribute name="IsPrimaryMVG" type="xsd:string" />
</xsd:complexType>
</xsd:complexType>

This will produce the following XML:

```xml
<hous:Household>
  <hous:ListOfRelatedContact>
    <hous:RelatedContact EDStartDate="1/1/2003" EDEndDate="">
      <hous:ContactId>88-30KSP</hous:ContactId>

      <hous:EDListOfFirstName>
        <hous:FirstName EDStartDate="04/01/2012" EDEndDate="">SF1N6</hous:FirstName>
      </hous:EDListOfFirstName>
      <hous:EDListOfLastName>
      </hous:EDListOfLastName>
    </hous:RelatedContact>
  </hous:ListOfRelatedContact>
</hous:Household>
```
About Siebel EAI Adapter Operations for Effective Dating on Links

The Siebel EAI Adapter receives the integration object in the format described in the "Web Service Schema Example" on page 182. Depending on the specified operations the effective dates are used as described in the following:

Insert Operation
After the record is inserted into the parent and child business component, the Siebel EAI Adapter reads the EDStartDate and the EDEndDate from the integration object and inserts these values into the corresponding effective dating business component.

Update Operation
After the record is inserted into the parent and child business component, the Siebel EAI Adapter removes all history records from the effective dating enabled business component and then reads the EDStartDate and EDEndDate values from the integration object and inserts these dates as fresh records into the business component.

NOTE: The Update operation is only possible for currently active links (in other words, update is not possible if a link has been soft deleted by giving a end date value for the most recent history record).

Upsert and Synchronize Operations
If the upsert attribute is specified for the operation in the integration component then no history records are deleted, instead the history given in the XML input is inserted into the history table.

The synchronize operation can result in the insert, update or deletion of the child integration component as specified in the conditions set in the "Synchronize Method" on page 150.
**XML Example**

This example illustrates what is required if you want to perform an insert or upsert operation to insert or update multiple history records for the same child business component.

**NOTE:** Multiple entries must be specified in the input XML code with same user key.

```xml
<hous:Household>
    
    <hous:ListOfRelatedContact>
        
        <hous:RelatedContact EDStartDate="1/1/2003" EDEndDate=""> 
            <hous:ContactId>88-30KSP</hous:ContactId>
        </hous:RelatedContact>
        
        <hous:RelatedContact EDStartDate="1/1/2002" EDEndDate="12/31/2002"> 
            <hous:ContactId>88-30KSP</hous:ContactId>
        </hous:RelatedContact>
        
        <hous:RelatedContact EDStartDate="1/1/2001" EDEndDate="12/31/2001"> 
            <hous:ContactId>88-30KSP</hous:ContactId>
        </hous:RelatedContact>
        
        <hous:RelatedContact EDStartDate="1/1/2000" EDEndDate="12/31/2000"> 
            <hous:ContactId>88-30KSP</hous:ContactId>
        </hous:RelatedContact>

    </hous:ListOfRelatedContact>

</hous:Household>
```

**About Using Language-Independent Code with the EAI Siebel Adapter Business Service**

If the user property AllLangIndependentVals is set to Y at the integration object level, then the EAI Siebel Adapter uses the language-independent code for its LOVs.

In the outbound direction, for example, the Query method, if the AllLangIndependentVals is set to Y, then the EAI Siebel Adapter translates the language-dependent values in the Siebel Database to their language-independent counterpart based on the List Of Values entries in the database.
In the inbound direction, for example the Synchronize method, if the AllLangIndependentVals is set to Y, then the EAI Siebel Adapter expects language-independent values in the input message, and translates them to language-dependent values based on the current language setting and the entries in the List Of Values in the database.

NOTE: The LOV-based fields are always validated using language-dependent values. Using language independent values for LOVs and MLOVs increases the EAI Siebel Adapter CPU usage by about five percent, but allows easier communication between systems that operate on different languages.

About LOV Translation and the EAI Siebel Adapter Business Service

The Siebel application distinguishes two types of lists of values (LOV):

- **Multilingual LOV (MLOV).** Stores a language-independent code (LIC) in the Siebel Database that is translated to a language-dependent value (LDV) for active language by Object Manager. MLOVs are distinguished by having the Translation Table specified in the Column definition.

- **Single-language LOV.** Stores the LDV for the current language in the Siebel Database. The Boolean integration object user property AllLangIndependentVals determines whether the EAI Siebel Adapter will use LDV (No = no translation necessary) or LIC (Yes = translation needed) for such LOVs.

Translating to LIC affects performance, but allows easier cooperation between systems that operate on different languages. This option is especially used by various import and export utilities.

The AllLangIndependentVals integration object user property is undefined for integration objects when the base object type is not Siebel Business Object. When the base object is Siebel Business Object, AllLangIndependentVals is defined with a default value of N.

Table 26 explains the behavior of the EAI Siebel Adapter according to the AllLangIndependentVals integration object user property values.

<table>
<thead>
<tr>
<th>AllLangIndependentVals</th>
<th>Yes</th>
<th>No</th>
<th>Undefined</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOV</td>
<td>LIC</td>
<td>LDV</td>
<td>LDV</td>
</tr>
<tr>
<td>MLOV</td>
<td>LIC</td>
<td>LDV</td>
<td>LIC</td>
</tr>
</tbody>
</table>

Siebel EAI and Run-Time Events

The Siebel application allows triggering workflows based on run-time events or workflow policies.

**Run-Time Events.** Siebel EAI supports triggering workflows based on run-time events such as Write Record, which is triggered whenever a record is written. If you use the EAI Siebel Adapter to import data into Siebel Business Applications, and use run-time events, then consider the following:
For the EAI Siebel Adapter, one call to the EAI Siebel Adapter with an input message is a transaction. Within a transaction, the EAI Siebel Adapter makes multiple Write Record calls. At any point in the transaction, if the EAI Siebel Adapter encounters a problem the transaction is rolled back entirely. However, if you have specified events to trigger at Write Record, such events are called as soon as the EAI Siebel Adapter makes Write Record calls even though the EAI Siebel Adapter might be in the middle of a transaction. If you have export data workflows triggered on such events, this might lead to exporting data from Siebel Business Applications that is not committed and might be rolled back. It is also possible that your events are triggered when the record is not completely populated, which leads to situations that are not handled by your specified event processing workflow.

To avoid the effects of this interaction between the EAI Siebel Adapter and run-time events use the business service EAI Transaction Service to figure out if a transaction (typically, the EAI Siebel Adapter) is in progress. You might then want to skip processing that is not desirable when the EAI Siebel Adapter is in progress.

For example, suppose you have a workflow to export orders from Siebel Business Applications, which is triggered whenever the order record is written. You also import orders into Siebel Business Applications using EAI. In such a situation, you do not want to export orders while they are being imported, because the import might be aborted and rolled back. You achieve this using the EAI Transaction Service business service as the first step of the export workflow. If you find that a transaction is in process you can branch directly to the end step.

**Workflow Policies.** In addition to Run-Time Events, Siebel Business 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 in this topic. Use Workflow Policies instead of Run-Time Events when possible.

---

**Guidelines for Using the EAI Siebel Adapter Business Service**

The following guidelines are to be considered when using the EAI Siebel Adapter:

- Keep the integration objects small. Basically, inactivate any unused fields in the integration component. Avoid creating large integration object instances.

- Test the developed object definitions using the EAI Siebel Adapter before adding to production. You must test your input and output using working and negative scenarios. Also do performance testing to make sure you are satisfied with the performance of the input and the output.

- Oracle does not support the use of EAI to update data that is based on administration-type business components such as Client - Mobile or Position. Only the System Administrator updates these types of data.

- Always use a search specification with the Query() method to avoid receiving every object when run.

- To optimize database performance, you can explicitly specify that the EAI Siebel Adapter use only user key fields. This feature is available for the methods Query, Delete, and Execute. To use it, set the input property QueryByUserKey to True for the EAI Siebel Adapter business service and pass an integration object instance (for example, a Siebel Message) as an input as well. By default, the Siebel adapter uses all the fields in the input integration object instance.
Troubleshooting the EAI Siebel Adapter Business Service

The EAI Siebel Adapter natively accesses Siebel objects definitions using the business objects, integration objects, and business component classes. Because of this design, you might get an EAI Siebel Adapter error that contains an error message from the Siebel Object Manager. See Figure 1 on page 16 for a logical overview of the Siebel architectural layers. Figure 1 on page 16 also shows the component events that will help you determine in which layer of the application the problem is occurring.

The EAI Siebel Adapter functionality must be considered in light of the entire application functionality. For example, the Siebel Communications product line provides preconfigured Asset Based Ordering functionality that uses Siebel workflows and business services. The workflows use the EAI Siebel Adapter business service to extract data from the database and to update the database.

When using this functionality, the possibility exists that you might get an error in a step of the workflow that indicates a problem with the EAI Siebel Adapter, such as the asset you want to insert already exists in the system. In this case, first verify that you are not inserting a duplicate asset. If you have validated that the asset is new and not a duplicate, then you must research the specifics as to why the EAI Siebel Adapter failed to insert the new asset or attempted to insert a duplicate asset.

If you have modified the preconfigured Asset integration object or business object, it could be one of your customizations. For example, perhaps your asset requires additional fields, and you are not providing those fields in your inbound integration object instance. Therefore, it uses any default values, thus creating a potential duplicate asset.

Enabling Logging for the EAI Siebel Adapter Business Service

Using component events, logging can be done in the Siebel application. Components are used to assist with the debugging of problems in the Siebel application. A list of useful and relevant component events for debugging EAI Siebel Adapter problems are listed in Table 27. These components events can be enabled on any server component that is capable of running an EAI process and on the Siebel client. You might want to enable other events not listed in Table 27.
For more information on how to perform logging in the Siebel clients, see *Siebel System Monitoring and Diagnostics Guide*.

Table 27. Component Events for Debugging EAI Siebel Adapter Problems

<table>
<thead>
<tr>
<th>Event Alias Name</th>
<th>Logging Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAISiebAdpt</td>
<td>4 or 5</td>
<td>Captures EAI Siebel Adapter related events, including integration component and integration component fields accessed and the values for the fields; business components and business component fields accessed and the values for the fields. This is the main event to enable for EAI Siebel Adapter troubleshooting.</td>
</tr>
<tr>
<td>EAISiebAdptPerf</td>
<td>4</td>
<td>Captures EAI Siebel Adapter performance related events, including operation performed and time for the operation in milliseconds. This event summarizes the result of the EAI Siebel Adapter operation. For more information on performance logging, see “Troubleshooting the EAI Siebel Adapter Business Service” on page 188 and Doc ID 476905.1 on My Oracle Support. This document was previously published as Siebel FAQ 1840.</td>
</tr>
<tr>
<td>EAISiebAdptSvcArgTrc</td>
<td>3 or 4</td>
<td>Dumps the inputs and output arguments for the EAI Siebel Adapter when EnableServiceArgTracing=true. For more information on argument tracing, see “Enabling Siebel Argument Tracing” on page 190.</td>
</tr>
<tr>
<td>EAITransaction</td>
<td>4</td>
<td>Captures when an EAI Transaction starts.</td>
</tr>
<tr>
<td>EAIInfra</td>
<td>4</td>
<td>Output Message: IntObjType=Contact Interface Format=Siebel Hierarchical</td>
</tr>
<tr>
<td>EAIQrySpec</td>
<td>4</td>
<td>Captures the search specification if one is specified.</td>
</tr>
<tr>
<td>SQL</td>
<td>4</td>
<td>Captures SQL executed against the database.</td>
</tr>
<tr>
<td>SQLParseAndExecute</td>
<td>4</td>
<td>Captures SQL statements and shows SQL bind parameters executed. Shows SQL executed against the database. May sometimes be different than the SQL show in ObjMgrSQLLog.</td>
</tr>
<tr>
<td>ObjMgrLog</td>
<td>4 or 5</td>
<td>Logs error code and error message encountered by various Siebel objects.</td>
</tr>
<tr>
<td>ObjMgrDataLog</td>
<td>4</td>
<td>Logs the beginning of a transaction for the database connection.</td>
</tr>
</tbody>
</table>
Enabling Siebel Argument Tracing

You can also export input and output arguments in XML format to a file for the EAI Siebel Adapter. These XML files represent the input and output arguments integration object instances. This is a useful technique as it writes to a file the integration object instances in the directory where your Siebel process is running. For example, in the Siebel Developer Web Client, it is `c:/siebel/bin`.

### To enable output arguments tracing

1. Set the server parameter EnableServiceArgTracing to True:
   - If you are running the Siebel Developer Web Client, then add the following to your .cfg file:
     ```
     [EAI Subsys]
     EnableServiceArgTracing = TRUE
     ```
   - If you are running the Siebel Web Client, modify the following Siebel Server parameter for your object manager:
Configuring the EAI Siebel Adapter Business Service for Concurrency Control

The EAI Siebel Adapter supports concurrency control to guarantee data integrity and avoid overriding data by simultaneous users or integration processes. To do so, the EAI Siebel Adapter uses the Integration Component Key called the Modification Key.

Modification Key

A Modification Key is an Integration Component Key of the type Modification Key. A Modification Key is a collection of fields that together are used to verify the version of an integration component instance. Typically, Modification Key fields are Mod Id fields for the tables used. Multiple Modification Key fields might be needed, because a business component might be updating multiple tables, either as extension tables, or through implicit or explicit joins.

The EAI Siebel Adapter methods (Insert, Update, Synchronize, Upsert) check for the existence of a Modification Key. If no Modification Key is specified in the integration component definition, or if Modification Key fields are not included in the XML request, then the EAI Siebel Adapter does not check for the record version and proceeds with the requested operation. If a valid Modification Key is found, but the corresponding record cannot be found, then the EAI Siebel Adapter assumes that the record has been deleted by other users and returns the error SSASqlErrWriteConflict.

If a valid Modification Key as well as the corresponding record can be found, then the EAI Siebel Adapter checks if the Modification Key fields in the XML request and the matched record are consistent. If any of the fields are inconsistent, then the EAI Siebel Adapter assumes that the record has been modified by other users and returns the error SSASqlErrWriteConflict. If all the fields are consistent, then the EAI Siebel Adapter proceeds with the requested operation.
Modification IDs

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

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

When using Modification IDs, the following behaviors are present:

- All fields must be present in the integration object instance for the Mod Key to be used.
- Only one defined Modification Key is present for each integration component. Unlike for User Keys, multiple Mod Keys are not allowed.

About the Modification ID for a Base Table

The integration component field Mod Id for a base table is created by the Integration Object Builder Wizard, but you must make sure it is active if it is needed for your business processes.

About the Modification ID for an Extension Table

An extension table’s Mod Id field is accessible as extension table name.Mod Id in the business component, for example, S_ORG_EXT_X.Mod Id. However, if your business processes require this field, then you must manually add it to the integration object definition by copying the Mod Id field and changing the properties.

About the Modification ID for a Joined Table

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

- Be prefixed with CX string and preferably followed by the name of the join
- Be Joined over the correct join
- Have MODIFICATION_NUM specified as underlying column of type DTYPE_INTEGER

About MVG and MVGAssociation Integration Components

For integration components that are of type MVG or MVGAssociation, in addition to the preceding steps, you must create user properties MVGFieldName and AssocFieldName for each Modification ID integration component field, respectively, and set the name of the field shown in the parent business component as the value.
To configure the EAI Siebel Adapter business service for concurrency control

1. For each integration component, identify all needed Modification IDs:

   **NOTE:** In addition to the Modification ID for the base table, Modification IDs for tables that are used through one-to-one extension as well as through implicit joins are relevant. For example, on modifying an account record MODIFICATION_NUM column on S_ORG_EXT is updated, not the MODIFICATION_NUM column on S_PARTY.

   a. Identify all active fields in an integration component that will be updated and have to be concurrency safe.

   b. Select the corresponding business component, the value in the External Name property of the integration component.

   c. For each field identified in Step a, check the value of the Join property of the field. If the join is not specified, then the field belongs to the base table; otherwise, note the name of the join.

   d. In the Object Explorer, select Business Component, then Join, and query for the business component from Step b. Search whether there is an entry whose Alias property matches the name of the join from Step c:
      - If a matching Alias is found, then this field belongs to a Joined Table. The name of the join in Step c is the join name, and the value of the Table property is the joined table.
      - If no Alias matches, then this is an implicit join to an Extension Table. The name of the join in Step c is the name of the extension table.

2. Create business component fields for Mod Ids of Joined Tables. For the previous example, create a new field in the business component Account with the following settings:

   - **Name.** CX_Primary Organization-S_BU.Mod Id
   - **Join.** Primary Organization-S_BU
   - **Column.** MODIFICATION_NUM
   - **Type.** DTYPE_INTEGER

3. Expose all Modification IDs identified in Step 1 as integration component fields.

4. For MVG and MVG Association integration components, add user property MVGFieldName and AssocFieldName respectively, on all Modification ID fields as follows:

   a. Check the Integration Component User Prop sub type for user properties of the integration component.

   b. If there is a user property called MVGAssociation, then the integration component is a MVG Association, but if there is a user property called Association then the integration component is a MVG.

      **NOTE:** If the integration component is neither an MVG nor an MVG Association, then nothing is required to be done.

5. Repeat the following steps for each Modification ID field on the integration component:

   a. Add user property MVGFieldName if MVG, or AssocFieldName if MVG Association.

   b. Set the value of the user property to the same as the field name, for example, Mod Id, extension table name.Mod Id, or CX_join.Mod Id.
6 Create Modification Key.

Define a new integration component key of type Modification Key, and include all the integration component fields exposed in Step 3 on page 193 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 must store the value of the Modification IDs when it queries data from the Siebel Database.
   b The client program must send exactly the same values of the Modification IDs that it retrieved from the Siebel Database when sending an update.
   c The client program must not send any Modification IDs when sending a new record to the Siebel application. If this is violated, then the client program generates an error indicating that the record has been deleted by another user.

Integration Component Account Example
Consider an integration component Account of the business component Account:
   ■ Field Home Page has property Join set to S_ORG_EXT. This is an implicit join, because it is not listed in the joins; therefore, this field belongs to Extension Table S_ORG_EXT.
   ■ Field Primary Organization has property Join set to Primary Organization-S_BU. This is an explicit join, because it is listed in the joins. The value of Table property is S_BU; therefore, this field belongs to Joined Table S_BU joined over Primary Organization-S_BU.

   Activate integration component field Mod Id:
   ■ Set Name, External Name, XML Tag properties to Mod Id
   ■ Set External Data Type property to DTYPE_NUMBER
   ■ Set External Length property to 30
   ■ Set Type property to System

   Add integration component field S_ORG_EXT.Mod Id:
   ■ Set Name, External Name, XML Tag properties to S_ORG_EXT.Mod Id
   ■ Set External Data Type property to DTYPE_NUMBER
   ■ Set External Length property to 30
   ■ Set Type property to System

   Add integration component field CX_Primary Organization-S_BU.Mod Id:
   ■ Set Name, External Name, XML Tag properties to CX_Primary Organization-S_BU.Mod Id
   ■ Set External Data Type property to DTYPE_NUMBER
   ■ Set External Length property to 30
   ■ Set Type property to System
Integration Component Account_Organization Example

Consider the integration component Account_Organization of the Sample Account integration object. Account_Organization is an MVG Association as denoted by the presence of the user property MVGAssociation. Assume two Modification IDs, Mod Id and S_ORG_EXT.Mod Id, were exposed on this integration component:

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

In this integration component example, Account of the Sample Account integration object, takes the following action:

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

Status IDs

When using Status IDs with Modification IDs, the following behavior can be present:

- All fields must be present in the integration object instance for the Modification Key to be used.
- Only one defined Modification Key is present for each integration component. Unlike User Keys, multiple Modification Keys are not used with Status IDs.
The EAI UI Data Adapter business service is used for exposing Siebel data to external user interfaces. This chapter includes the following topics:

- About the EAI UI Data Adapter Business Service on page 197
- EAI UI Data Adapter Business Service Methods on page 199
- EAI UI Data Adapter Business Service Method Arguments on page 215
- Exposing Siebel Data to External Applications on page 216

About the EAI UI Data Adapter Business Service

The EAI UI Data Adapter business service exposes an interface with weakly typed arguments that can query and update data in the Siebel database. The EAI UI Data Adapter service is called indirectly by UI Data Sync Services, which are published externally as Web services.

The EAI UI Data Adapter is similar to the EAI Siebel Adapter business service, but contains key differences that make it more suitable for UI rendering by custom Web applications. The differences are summarized as follows:

- **Row Id as User Key.** Unlike the EAI Siebel Adapter, the EAI UI Data Adapter does not use user keys defined in the integration object. It uses an implicit, hard-coded user key, which comprises the Row Id field.

  For more information about how User Keys are used with the EAI Siebel Adapter, see “About Integration Component Keys” on page 32.

- **Row Id and Mod Id as Status Key.** Unlike the EAI Siebel Adapter, the EAI UI Data Adapter does not use status keys defined in the integration object. It uses an implicit, hard-coded status key, which comprises the fields Row Id and Mod Id.

  For more information about how Status Keys are used with the EAI Siebel Adapter, see “About Integration Component Keys” on page 32.
Operation Semantics on Leaf Nodes. In an integration object hierarchy, nodes with at least one child are called internal nodes and nodes without children are called leaf nodes. When either the insert or update method is called on the EAI Siebel Adapter, the adapter performs the operation on both internal nodes and leaf nodes. When the insert or update method is called on the EAI UI Data Adapter, the adapter performs insert on leaf nodes only as represented in Figure 34.

![Figure 34. Operation Semantics on Leaf Nodes](image)

The match nodes in Figure 34 reflects that the database contains a record with the same user keys as the integration object instance.

Predefined Queries. The EAI UI Data Adapter extends the Query Page functionality of the EAI Siebel Adapter. The EAI UI Data Adapter can take the name of a predefined query and execute the query.

For detailed information about the QueryPage method, see “QueryPage Method” on page 199.

Child Pagination. The EAI UI Data Adapter supports child pagination, enabling custom UIs to render one page of data at a time.

For more information, see “Root and Child Pagination” on page 200.

Strongly Typed Data. Unlike the EAI Siebel Adapter, the EAI UI Data Adapter supports the exchange of strongly typed data.

The EAI UI Data Adapter is most suitable for use in custom UI development where the service is called indirectly by Web services. In other types of integration scenarios, the EAI Siebel Adapter is a more suitable choice. For more information about the EAI Siebel Adapter, see “EAI Siebel Adapter Business Service” on page 145.
EAI UI Data Adapter Business Service Methods

The EAI UI Data Adapter service provides access to the following methods:

- "QueryPage Method" on page 199
- "UpdateLeaves Method" on page 204
- "InitLeaves Method" on page 206
- "InsertLeaves Method" on page 208
- "DeleteLeaves Method" on page 212
- "Execute Method" on page 213

QueryPage Method

Custom UIs can use the QueryPage method to query data in the Siebel database one page at a time. QueryPage supports both query-by-example (QBE) and predefined queries (PDQ). However, it is recommended that you use either QBE or a PDQ, but not both at the same time. If both QBE and PDQ are specified, then PDQ overrides QBE. In this case, the EAI UI Data Adapter executes the PDQ, ignores the QBE, and does not generate an error.

QueryPage Method Arguments

Table 28 lists the method arguments used with the QueryPage method. For a description of the arguments, see “EAI UI Data Adapter Business Service Method Arguments” on page 215.

Table 28. Method Arguments for QueryPage

<table>
<thead>
<tr>
<th>Method Argument Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExecutionMode</td>
<td>Input</td>
</tr>
<tr>
<td>LOVLanguageMode</td>
<td>Input</td>
</tr>
<tr>
<td>NamedSearchSpec</td>
<td>Input</td>
</tr>
<tr>
<td>NewQuery</td>
<td>Input</td>
</tr>
<tr>
<td>NumOutputObjects</td>
<td>Output</td>
</tr>
<tr>
<td>OutputIntObjectName</td>
<td>Input</td>
</tr>
<tr>
<td>SiebelMessage</td>
<td>Input / Output</td>
</tr>
<tr>
<td>ViewMode</td>
<td>Input</td>
</tr>
</tbody>
</table>
Root and Child Pagination

The EAI UI Data Adapter supports pagination for both root and child components. To support root and child pagination, the EAI UI Data Adapter requires that you set the attributes listed in Table 29 as part of the integration component instance.

**NOTE:** Pagination over root components benefits performance because, as long as the search specification, sort specification, and view mode remain the same, the business component is not re-executed with each invocation of QueryPage. However, for pagination over child components, the component is reexecuted every time you call QueryPage.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pagesize</td>
<td>The number of records to be returned for a component. The default page size is 10. Note that there is a server parameter that controls the maximum page size (MaximumPageSize). If the pagesize attribute is greater than the maximum pagesize defined in the server parameter, then an error occurs.</td>
</tr>
<tr>
<td>startrownum</td>
<td>Determines the starting point for record retrieval. The 0-based index of the record within the recordset.</td>
</tr>
<tr>
<td>lastpage</td>
<td>Indicates whether the record being returned is the last record in the record set. The value is set by the EAI UI Data Adapter. Valid values are true or false.</td>
</tr>
<tr>
<td>recordcountneeded</td>
<td>When set to true, indicates that a record count is needed for this component. Valid values are true or false.</td>
</tr>
<tr>
<td>recordcount</td>
<td>Value set by the EAI UI Data Adapter indicating the approximate record count provided by the object manager based on the search specification.</td>
</tr>
</tbody>
</table>

Example of QueryPage on Parent and Child Components

This example demonstrates querying on both parent and child components. In this example, the query is for accounts that begin with ‘A’ and any associated contacts (First Name and Last Name). Note that pagesize is 10 and an approximate record count is requested and returned in the response.

Request

```xml
<SiebelMessage MessageType='Integration Object' IntObjectName='Account'
IntObjectFormat='Siebel Hierarchical'>
<ListOfAccount pagesize='10' startrownum='0' recordcountneeded='true'>
  <Account>
    <Name>='A'</Name>
    <ListOfContact>
      <Contact>
        <FirstName></FirstName>
        <LastName></LastName>
      </Contact>
    </ListOfContact>
  </Account>
</ListOfAccount>
</SiebelMessage>
```
Response
Siebel Message MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
<ListOfAccount recordcount="2" lastpage="true">
<Account>
  <Name>Adams Tech</Name>
  <ListOfContact lastpage="true">
    <Contact>
      <FirstName>Sally</FirstName>
      <LastName>Brown</LastName>
    </Contact>
    <Contact>
      <FirstName>Terry</FirstName>
      <LastName>Smith</LastName>
    </Contact>
  </ListOfContact>
</Account>
<Account>
  <Name>Aleph Inc.</Name>
  <ListOfContact lastpage="true">
    <Contact>
      <FirstName>Bill</FirstName>
      <LastName>Jones</LastName>
    </Contact>
    <Contact>
      <FirstName>Roland</FirstName>
      <LastName>Smith</LastName>
    </Contact>
  </ListOfContact>
</Account>
</ListOfAccount>
</SiebelMessage>
Sort Specification
You can specify a sort specification on one or more integration component fields of an integration component. For each field you want sort on, you must define the attributes listed in Table 30. If both attributes are not specified, then the field is not considered as a sort field.

Table 30. Sort Specification Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sortorder</td>
<td>Determines whether the sort order is ascending or descending. Valid values are ASC or DEC.</td>
</tr>
<tr>
<td>sortsequence</td>
<td>Determines the order in which the sort specification is applied. Valid values are integer numbers.</td>
</tr>
</tbody>
</table>

Example of Sort Specification
This example demonstrates using the QueryPage method with an ascending sort order.

Request
```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Row_Id>2-1111</Row_Id>
      <ListOfContact pagesize="40" startrownum="0" recordcountneeded="true">
        <Contact>
          <FirstName sortorder="ASC" sortsequence="1">Alice</FirstName>
        </Contact>
        <Contact>
          <FirstName>Bill</FirstName>
        </Contact>
        <Contact>
          <FirstName>Casey</FirstName>
        </Contact>
      </ListOfContact>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```

Response
```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount lastpage="true">
    <Account>
      <Row_Id>2-1111</Row_Id>
      <ListOfContact recordcount="3" lastpage="true">
        <Contact>
          <FirstName>Alice</FirstName>
        </Contact>
        <Contact>
          <FirstName>Bill</FirstName>
        </Contact>
        <Contact>
          <FirstName>Casey</FirstName>
        </Contact>
      </ListOfContact>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```
Predefined Query
You can specify the name of a PDQ using the method argument `NamedSearchSpec`. The EAI UI Data Adapter uses this value to set the search specification at the business object level.

Search Specification
You can use the searchspec attribute on a component instance for complicated queries.

For example, query by example (QBE) uses AND as the implicit operator between fields. You could implement OR semantics by using multiple integration component instances, but this would result in a query for each integration component instance and might result in duplicate records being returned. Using the searchspec attribute could avoid this problem.

The syntax for the searchspec attribute is as follows:

- Expression: `Expression [Binary Operator Expression]`
- Expression: `[Field XML tag] Operator 'Value'
- Expression: `(Expression)`
  
  **NOTE:** Parentheses can be nested.

- Expression: `[Field XML tag] IS NULL | [Field XML tag] IS NOT NULL
- Expression: `EXISTS(Expression) | NOT EXISTS(Expression)`
  
  **NOTE:** In EXISTS and NOT EXISTS expressions, use the business component field names of multivalue group (MVG) fields, not the integration component XML tag names.

- Operator: `= | ~= | < | <= | > | >= | <> | LIKE | ~LIKE`
- Binary Operator: AND | OR

The EAI UI Data Adapter parses the searchspec (unlike the EAI Siebel Adapter) and performs the following operations before setting the search specification on the business component:

- Converts Field XML tags into business component field names. For example, assume two business component fields, First Name and Last Name, have XML tags `FirstName` and `LastName` respectively. The EAI UI Data Adapter converts the XML tags as shown in Table 31.

### Table 31. Example Search Specification Conversion

<table>
<thead>
<tr>
<th>This Search Spec</th>
<th>Will be converted to this</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[FirstName] LIKE '*Jon*' AND [LastName] = 'Doe'</code></td>
<td><code>[FirstName] LIKE '*Jon*' AND [Last Name] = 'Doe'</code></td>
</tr>
<tr>
<td><code>[FirstName] LIKE '*Jon*' OR [LastName] LIKE 'Doe'*</code></td>
<td><code>[FirstName] LIKE '*Jon*' OR [Last Name] LIKE 'Doe'</code></td>
</tr>
</tbody>
</table>
If the input argument LOVLanguageMode is set to LIC, then LOV values are converted to language dependent codes. See "EAI UI Data Adapter Business Service Method Arguments" on page 215.

Validates operators, binary operators, and the syntax of the searchspec.

For more information about query language, see Siebel Developer’s Reference.

Example of Using the searchspec Attribute

This example demonstrates using the searchspec attribute for the QueryPage method.

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Id>2-1111</Id>
      <ListOfContact pagesize="10" startrownum="0">
        <Contact searchspec="[FirstName] LIKE '*Jon*' AND [LastName] = 'Doe'">
          <FirstName></FirstName>
          <LastName></LastName>
        </Contact>
      </ListOfContact>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```

UpdateLeaves Method

Use UpdateLeaves to update existing records in the Siebel database. When UpdateLeaves is called on an integration object hierarchy, the EAI UI Data Adapter updates leaf nodes only and uses internal nodes for maintaining parent-child relationships.

Both internal nodes and leaf nodes must have Row Ids specified or the EAI UI Data Adapter generates an error. The EAI UI Data Adapter also generates an error if it does not find a match for the internal node and leaf node for a given Row Id.

UpdateLeaves Method Arguments

Table 32 lists the method arguments used with UpdateLeaves. For a complete description of the method arguments, see “EAI UI Data Adapter Business Service Method Arguments” on page 215.

<table>
<thead>
<tr>
<th>Method Argument Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusObjCacheSize</td>
<td>Input</td>
</tr>
<tr>
<td>LOVLanguageMode</td>
<td>Input</td>
</tr>
<tr>
<td>SiebelMessage</td>
<td>Input / Output</td>
</tr>
</tbody>
</table>
Example of Updating Root Component
The following example demonstrates updating a root component.

Request
The following is an example of a request:

```xml
<SiebelMessage MessageType="Integration Object" IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>1-1-1111</Account>
    <Employees>4900</Employees>
  </ListOfAccount>
</SiebelMessage>
```

Response
The following is an example of a response:

```xml
<SiebelMessage MessageId="P-3ITT" MessageType="Integration Object" IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Id>1-1-1111</Id>
      <Mod_Id>2</Mod_Id>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```

Example of Updating Child Component
This example demonstrates updating a child component.

Request
The following is an example of a request:

```xml
<SiebelMessage MessageType="Integration Object" IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Id>1-1-1111</Id>
      <Employees>5000</Employees>
      <ListOfBusiness_Address>
        <Business_Address>
          <Id>2-2-2222</Id>
          <Postal_Code>94404</Postal_Code>
        </Business_Address>
      </ListOfBusiness_Address>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```
Response
The following is an example of a response:

```xml
<SiebelMessage MessageId="P-3ITW" MessageType="Integration Object" IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
    <ListOfAccount>
        <Account>
            <Id>1-1-1111</Id>
            <Mod_Id>2</Mod_Id>
            <ListOfAccount_Business_Address>
                <Business_Address>
                    <Id>2-2-2222</Id>
                    <Mod_Id>2</Mod_Id>
                </Business_Address>
            </ListOfAccount_Business_Address>
        </Account>
    </ListOfAccount>
</SiebelMessage>
```

InitLeaves Method

Use InitLeaves to retrieve pre-default values. When InitLeaves is called on an integration object hierarchy, it retrieves the pre-default values for all leaf nodes. All internal nodes must exist in the database and Row Id must be specified.

InitLeaves Method Arguments

Table 33 lists the method arguments used with the InitLeaves Method. For a complete description of the method arguments, see “EAI UI Data Adapter Business Service Method Arguments” on page 215.

<table>
<thead>
<tr>
<th>Method Argument</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusObjCacheSize</td>
<td>Input</td>
</tr>
<tr>
<td>LOVLanguageMode</td>
<td>Input</td>
</tr>
<tr>
<td>SiebelMessage</td>
<td>Input / Output</td>
</tr>
<tr>
<td>ViewMode</td>
<td>Input</td>
</tr>
</tbody>
</table>

Example of Using InitLeaves on a Root Component

The following code snippet demonstrates using InitLeaves to retrieve default values for a root component. In this example the root component is Account.

Request

The following is an example of a request:
<?xml version="1.0" encoding="UTF-8"?>
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Currency_Code></Currency_Code>
      <Account_Status></Account_Status>
      <Location_Type></Location_Type>
    </Account>
  </ListOfAccount>
</SiebelMessage>

Response
The following is an example of a response:

<?xml version="1.0" encoding="UTF-8"?>
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Account_Status>Active</Account_Status>
      <Currency_Code>USD</Currency_Code>
      <Location_Type>Corporate Training Center</Location_Type>
    </Account>
  </ListOfAccount>
</SiebelMessage>

Example of Using InitLeaves on a Child Component
The following code snippets demonstrate using InitLeaves to retrieve pre-default values for a child component. In this example the parent component is Account and the child component is Business Address.

Request
The following is an example of a request:

<?xml version="1.0" encoding="UTF-8"?>
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Id>1-111112</Id>
      <ListOfBusiness_Address>
        <Business_Address>
          <Active_Status></Active_Status>
          <Main_Address_Flag></Main_Address_Flag>
        </Business_Address>
      </ListOfBusiness_Address>
    </Account>
  </ListOfAccount>
</SiebelMessage>
Response
The following is an example of a response:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <ListOfBusiness_Address>
        <Business_Address>
          <Active_Status>Y</Active_Status>
          <Main_Address_Flag>Y</Main_Address_Flag>
        </Business_Address>
      </ListOfBusiness_Address>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```

InsertLeaves Method

Use InsertLeaves to insert records into the Siebel database. When InsertLeaves is called on an integration object hierarchy, the EAI UI Data Adapter inserts leaf nodes only and uses internal nodes for maintaining parent-child relationships:

- **Internal Nodes.** All internal nodes must already exist in the database and Row Id must be specified (Row Id is the implicit, hard-coded user key used by the EAI UI Data Adapter). If the internal node does not exist or Row Id is not specified, then the EAI UI Data Adapter returns an error. For more information about user keys, see “About the EAI UI Data Adapter Business Service” on page 197.

- **Leaf Nodes.** Whether or not Row Id must be specified for leaf nodes depends on the type of integration component:
  - If the integration component represents a normal business component or MVG business component, Row Id must not be defined, because records for these components are being inserted.
  - If the integration component represents an association business component or an MVG association business component, leaf nodes might or might not have Row Ids defined. If Row Ids are specified, then the EAI UI Data Adapter creates an association record only. If Row Ids are not specified, then both a child record and an association record are created.

InsertLeaves returns an integration object hierarchy. Each integration component instance in the hierarchy has two fields: Row Id and Mod Id (implicit status keys used by the EAI UI Data Adapter). You can use these fields to retrieve the Row Id of the newly created record.
InsertLeaves Method Arguments
Table 34 lists the method arguments used with the InsertLeaves method. For descriptions of the methods, see “EAI UI Data Adapter Business Service Method Arguments” on page 215.

Table 34. Method Arguments for InsertLeaves

<table>
<thead>
<tr>
<th>Method Argument Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusObjCacheSize</td>
<td>Input</td>
</tr>
<tr>
<td>LOVLanguageMode</td>
<td>Input</td>
</tr>
<tr>
<td>SiebelMessage</td>
<td>Input / Output</td>
</tr>
</tbody>
</table>

Example of Inserting a Root Component
This example code snippet demonstrates inserting a non-existing account.

Request
The following is an example of a request:

```xml
<SiebelMessage MessageType="Integration Object" IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Type>Competitor</Type>
      <Name>Dixon Inc.</Name>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```

Response
The following is an example of a response:

```xml
<SiebelMessage MessageId="P-3ITI" MessageType="Integration Object" IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Id>P-5NA84</Id>
      <Mod_Id>0</Mod_Id>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```
**Example of Inserting a Child Component**

The code snippets in this example demonstrate inserting a non-existing business address for an existing account.

**Request**

The following is an example of a request:

```xml
<SiebelMessage MessageType="Integration Object" IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Id>P-5NA84</Id>
      <ListOfBusiness_Address>
        <Business_Address>
          <City>San Carlos</City>
          <Street_Address>1145 laurel street</Street_Address>
          <State>CA</State>
          <Country>USA</Country>
          <Postal_Code>94063</Postal_Code>
        </Business_Address>
      </ListOfBusiness_Address>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```

**Response**

The following is an example of a response:

```xml
<SiebelMessage MessageId="P-3ITJ" MessageType="Integration Object" IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Id>P-5NA84</Id>
      <Mod_Id>1</Mod_Id>
      <ListOfBusiness_Address>
        <Business_Address>
          <Id>P-5NA8B</Id>
          <Mod_Id>0</Mod_Id>
        </Business_Address>
      </ListOfBusiness_Address>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```
Example of Inserting an Association Child Component

This example demonstrates inserting an existing organization for an existing account. This operation associates the organization with the account. If the organization does not exist, then the EAI UI Data Adapter generates an error.

Request

The following is an example of a request:

```xml
<SiebelMessage MessageType="Integration Object" IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Id>P-5NA84</Id>
      <ListOfAccount_Organization>
        <Account_Organization>
          <Id>1-123</Id>
        </Account_Organization>
      </ListOfAccount_Organization>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```

Response

The following is an example of a response:

```xml
<SiebelMessage MessageId="P-3ITL" MessageType="Integration Object" IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Id>P-5NA84</Id>
      <Mod_Id>1</Mod_Id>
      <ListOfAccount_Organization>
        <Account_Organization IsPrimaryMVG="Y">
          <Id>0-R9NH</Id>
          <ModId>9</ModId>
        </Account_Organization>
        <Account_Organization IsPrimaryMVG="N">
          <Id>1-123</Id>
          <ModId>0</ModId>
        </Account_Organization>
      </ListOfAccount_Organization>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```
DeleteLeaves Method

The DeleteLeaves method deletes leaf nodes only. If the Cascade Delete on the Link object is set to TRUE, then child records are also deleted. Row Ids are required for both internal nodes and leaf nodes. DeleteLeaves does not return a value when the operation is successful.

Method Arguments for DeleteLeaves

Table 35 lists the method arguments used with DeleteLeaves. For descriptions of the arguments, see “EAI UI Data Adapter Business Service Method Arguments” on page 215.

Table 35. Method Arguments for DeleteLeaves

<table>
<thead>
<tr>
<th>Method Argument Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntObjectName</td>
<td>Input</td>
</tr>
<tr>
<td>LOVLanguageMode</td>
<td>Input</td>
</tr>
<tr>
<td>SiebelMessage</td>
<td>Input / Output</td>
</tr>
</tbody>
</table>

Example of Deleting a Root Component

This example demonstrates deleting a root component.

```xml
<SiebelMessage MessageType="Integration Object" IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Id>P-5NA84</Id>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```

Example of Deleting a Child Component

This example demonstrates deleting a child component.

```xml
<SiebelMessage MessageType="Integration Object" IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Id>P-5NA84</Id>
      <ListOfBusiness_Address>
        <Business_Address>
          <Id>P-5NABA</Id>
        </Business_Address>
      </ListOfBusiness_Address>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```
Execute Method

The Execute method allows you to perform multiple operations on multiple business components. It is the only method that operates on internal nodes. The Execute method returns the same kind of object that the InsertLeaves method returns. For more information, see “InsertLeaves Method” on page 208.

**NOTE:** the Execute method requires a status object only when it contains an insert operation on a child integration component instance. However, because the EAI UI Data Adapter processes in a top-down fashion, it adds a status object to the integration object instance even if an insert operation is not present.

The operations are defined by the *operation* attribute on the integration component instance. An integration component instance can have the following operations as defined in Table 36.

### Table 36. Operation Attributes for Execute Method

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>update</td>
<td>Updates the integration component instance</td>
</tr>
<tr>
<td>insert</td>
<td>Inserts the integration component instance</td>
</tr>
<tr>
<td>delete</td>
<td>Deletes the integration component instance</td>
</tr>
<tr>
<td>skipnode</td>
<td>Matches integration component instances and process children</td>
</tr>
</tbody>
</table>

**CAUTION:** Operations must be specified on every integration component instance. If an operation is not specified, then an implicit Synchronize operation will be performed, which will delete all unmatched child integration component instances.

**Execute Method Arguments**

Table 37 lists the method arguments used with the Execute method. For a description of the methods, see “EAI UI Data Adapter Business Service Method Arguments” on page 215.

### Table 37. Method Arguments for Execute

<table>
<thead>
<tr>
<th>Method Argument Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusObjCacheSize</td>
<td>Input</td>
</tr>
<tr>
<td>LOVLanguageMode</td>
<td>Input</td>
</tr>
<tr>
<td>SiebelMessage</td>
<td>Input / Output</td>
</tr>
</tbody>
</table>

**Example of Using the Execute Method**

The following example demonstrates using the Execute method to perform update, insert, and delete operations on child object. Note that the skipnode operation is defined on the parent object.
**Request**

The following is an example of a request:

```xml
<SiebelMessage MessageType="Integration Object" IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
    <List Of Account>
        <Account operation="skipnode">
            <Id>1-1-1111</Id>
            <List Of Business_Address>
                <Business_Address operation="update">
                    <Id>2-2-2222</Id>
                    <Postal_Code>94402</Postal_Code> <!--Postal Code changed-->
                </Business_Address>
                <Business_Address operation="insert">
                    <Postal_Code>94402</Postal_Code>
                    <City>San Mateo</City>
                    <Street_Address>2215 Bridgepointe Parkway</Street_Address>
                    <State>CA</State>
                    <Country>USA</Country>
                </Business_Address>
                <List Of Contact>
                    <Contact operation="delete">
                        <Id>4-4-4444</Id>
                    </Contact>
                </List Of Contact>
            </List Of Business_Address>
        </Account>
    </List Of Account>
</SiebelMessage>
```

**Response**

The following is an example of a response:

```xml
<SiebelMessage MessageId="42-21YQ" MessageType="Integration Object" IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
    <List Of Account>
        <Account>
            <Id>1-1-1111</Id>
            <Mod_Id>3</Mod_Id>
            <List Of Business_Address>
                <Business_Address>
                    <Id>2-2-2222</Id>
                    <Mod_Id>1</Mod_Id>
                </Business_Address>
                <Business_Address>
                    <Id>42-53Q2W</Id>
                    <Mod_Id>0</Mod_Id>
                </Business_Address>
            </List Of Business_Address>
        </Account>
    </List Of Account>
</SiebelMessage>
```
EAI UI Data Adapter Business Service Method Arguments

The methods exposed in the EAI UI Data Adapter business service take arguments that you use to specify information that the adapter uses when processing the request. Table 38 summarizes these method arguments.

### Table 38. EAI UI Data Adapter Business Service Method Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Display Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusObjCacheSize</td>
<td>Business Object Cache Size</td>
<td>Maximum Number of Business Objects that can be cached at one time.</td>
</tr>
<tr>
<td>ExecutionMode</td>
<td>Execution Mode</td>
<td>Used to set the direction of a query on a business component. Valid values are ForwardOnly and Bidirectional. The default is Bidirectional.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ForwardOnly is more efficient than Bidirectional, and is recommended in cases where you must process a large number of records in the forward direction only (such as for report generation). For operations that are likely to return more than 10000 records, use ForwardOnly to avoid errors. For more information on executing queries, see the topic on the ExecuteQuery business component method in Siebel Object Interfaces Reference.</td>
</tr>
<tr>
<td>LOVLanguageMode</td>
<td>LOV Language Mode</td>
<td>Indicates whether the EAI UI Data Adapter must translate the LOV value before sending it to the object manager. Valid values are LIC or LDC. If LIC is specified, then the EAI UI Data Adapter expects language independent values in the input message and translates them to language dependent values (based on the current language setting) before the request is sent to the object manager. If LDC is specified, then the EAI UI Data Adapter does not translate the value before sending it to the object manager.</td>
</tr>
<tr>
<td>NamedSearchSpec</td>
<td>Predefined Query</td>
<td>Name of a PDQ. The EAI UI Data Adapter sets the name of the PDQ on the business object instance. If NamedSearchSpec and QBE are specified, then NamedSearchSpec is used.</td>
</tr>
<tr>
<td>NewQuery</td>
<td>New Query</td>
<td>Default is False. Boolean indicating whether a new query will be executed. If set to True, then a new query is executed flushing the cache for that particular integration object.</td>
</tr>
</tbody>
</table>
Exposing Siebel Data to External Applications

UI Data Sync (UDS) Web services provide external Web applications access to Siebel data and functionality. UDS services expose version-independent interfaces with strongly typed arguments and provide access to UI operations such as insert, delete, update, and query.

Some UDS Web services have already been preconfigured in Siebel Tools. For information on preconfigured UDS services, see *Siebel Web UI Dynamic Developer Kit Guide*.

The following procedure details how to reuse an integration object with the Data Access Service wizard in Siebel Tools to create a new UDS Web service and deploy it to the Siebel run-time database.

**NOTE:** You can also create a new integration object in this wizard, in which case the Integration Object Builder wizard is also started. For more information on using the Integration Object Builder wizard, see “Creating Integration Objects Using the EAI Siebel Wizard Business Service” on page 45.

**To make Siebel data accessible to external applications**

1. In Siebel Tools, create a new project and lock the project, or lock an existing project.
2. From the File menu, choose New Object to display the New Object Wizards dialog box.
3. Click the EAI tab, and then double-click Data Access Service.

   The Integration Object Page of the Data Access Service wizard appears.

   a. Select the locked project in which you want the business service and integration object to be.
   b. Choose whether to create a new integration object or reuse one.

   If you choose to create a new one, then the Integration Object Builder wizard will start when you click Next.
If you choose to reuse an integration object, then select it from the drop-down menu. Click Next.

4 On the Business Service Page:
   a Name the new business service.
      The UI Data Service is the only available base service.
   b (Optional) Select the Deploy Business Service check box to deploy the business service to the run-time database.
      Deployed business services are shown in the Administration - Business Services screen in the Siebel client.
   c Click Next.

5 On the UDS Methods Page, choose the methods to be exposed on the business service, and then click Next.
   The choices are the following:
   ■ Insert
   ■ Update
   ■ QueryPage
   ■ Delete
   ■ Execute
   ■ Init

6 On the Web Service Page:
   a In the URL for Web field, replace <webserver> with a valid host name and <lang> with a valid language code, such as enu.
   b Click Browse to choose a location to save the WSDL file.
   c Click Next.

7 (Optional) On the Report page:
   a Expand the integration object and integration components, and then select an integration component field to display the associated business component.
   b Select UI Business.
   c Click Browse to choose a location to save the exported report file.

8 Click Next.

9 (Optional) On the Integration Object Builder page, select the Deploy the Integration Object checkbox to deploy the integration object to the run-time database.
   NOTE: If you deploy integration objects while the Siebel Server is running, then you must subsequently clear the Web services cache in the Administration - Web Services screen, Inbound (or Outbound) Web Services view.

10 Examine the messages, then click Finish.
The business service, Web service, and integration object (if new) are created.
This chapter describes the virtual business component (VBC), its uses, and restrictions. This chapter also describes how you can create a new VBC in Siebel Tools. The following topics are included:

- About Virtual Business Components on page 219
- Using Virtual Business Components on page 222
- XML Gateway Service on page 224
- Examples of the Outgoing XML Format on page 227
- Search-Spec Node-Type Values on page 230
- Examples of the Incoming XML Format on page 231
- External Application Setup on page 234
- Custom Business Service Methods on page 234
- Custom Business Service Examples on page 250

### About Virtual Business Components

A virtual business component (VBC) provides a way to access data that resides in an external data source using a Siebel business component. The VBC does not map to an underlying table in the Siebel Database. You create a new VBC in Siebel Tools 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 VBCs, as illustrated in Figure 35 on page 220:

- Use the XML Gateway business service to pass data between the VBC and one of the Siebel transports, such as the EAI HTTP Transport or the EAI MSMQ Transport.
Write your own business service in Siebel eScript or in Siebel VB to implement the methods described in this chapter.

Using VBCs for Your Business Requirements

The following features enhance the functionality of VBCs to better assist you in meeting your business requirements:

- VBCs support drilldown behavior:
  - You can drill down on a VBC to a standard business component, another VBC, or the same VBC.
  - You can drill down onto a VBC from a standard business component, another VBC, or the same VBC.

- A parent applet can be based on a VBC.

- You can define VBCs that can participate as a parent in a business object. The VBC you define can be a parent to a standard BC or a VBC.

- You still can use an older version of the XML format or property set by setting the VBC Compatibility Mode parameter to the appropriate version. For information, see Table 39 on page 222.

- You can pass search and sort specifications to the business service used by a VBC.

- You can use the Validation, Pre Default Value, Post Default Value, Link Specification, and No Copy attributes of the VBC fields.
You can use predefined queries with VBCs.

You can have picklists based on VBCs, and use picklist properties such as No Insert, No Delete, No Update, No Merge, Search Specification, and Sort Specification.

You can use the Cascade Delete, Search Spec, Sort Spec, No Insert, No Update, and No Delete link properties when a VBC is the child business component on the link.

You can use the No Insert, No Update, No Delete, Search Spec, Sort Spec, and Maximum Cursor Size business component properties.

Usage and Restrictions for Virtual Business Components

The following are the uses and restrictions of VBCs:

- You can define a business object as containing both standard business components and VBCs.
- When configuring applets based on VBCs, use CSSFrame (Form) and CSSFrameList (List) instead of specialized applet classes.
- (Optional) Using the same name for the VBC field names and the remote data source field names can reduce the amount of required programming.
- VBCs cannot be docked, so they do not apply to remote users.
- VBCs cannot contain a multivalue group (MVG).
- VBCs do not support many-to-many relationships.
- A pick applet based on a VBC instantiates the VBC without any parent reference (no link is used). As result, the VBC business service does not receive the source field value from the parent component If the VBC business service must access the current parent business component context, then you can use the ActiveBusObject method of the TheApplication object in a business service server script to do the following:
  - Obtain the current business object instance (assuming this is the instance with this VBC).
  - Instantiate the parent business component (assuming the name of the parent BC is known).
  - Obtain the parent business component field for referencing it as a source (the field must be active in the current parent business component).
- VBCs cannot be loaded using Enterprise Integration Manager.
- Standard business components cannot contain multivalue groups based on VBCs.
- VBCs cannot be implemented using any business component class other than CSSBCVExtern. This means specialized business components such as Quotes and Forecasts cannot be implemented as VBCs.
- You cannot use Workflow Monitor to monitor VBCs.
- You cannot execute queries against VBCs when the search specification uses a function that is normally supported for Query mode against regular business components, such as ParentFieldValue().
Using Virtual Business Components

To use VBCs to share data with an external application, perform the following high-level tasks:

- "Creating a New Virtual Business Component" on page 222
- “Setting User Properties for the Virtual Business Component” on page 222
- Configuring your VBC business service:
  - Configure your XML Gateway Service or write your own business service.
    For information, see “XML Gateway Service” on page 224 and “Custom Business Service Methods” on page 234.
  - Configure your external application.
    For information, see “External Application Setup” on page 234.

Creating a New Virtual Business Component

You create a new VBC in Siebel Tools.

To create a new virtual business component

1. In Siebel Tools, lock the appropriate project.
2. In the Object Explorer, select the Business Component object.
3. Right-click, and then choose New Record.
4. Name the business component.
5. Select the project you locked in Step 1.
6. Set the class to CSSBCVExtern. This class provides the VBC functionality.

Setting User Properties for the Virtual Business Component

When defining the VBC, you must provide the user properties shown in Table 39.

Table 39. Setting Virtual Business Component User Properties

<table>
<thead>
<tr>
<th>User Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Name</td>
<td>The name of the business service.</td>
</tr>
<tr>
<td>Service Parameters</td>
<td>(Optional) Any parameters required by the business service. The Siebel application passes this user property, as an input argument, to the business service.</td>
</tr>
</tbody>
</table>
To define user properties for a virtual business component

1. In the Object List Editor in Siebel Tools, select the virtual business component for which you want to define user properties.

2. In the Object Explorer, expand the Business Component tree, and then select Business Component User Prop.

3. In the Object List Editor, click in the Business Component User Props list, right-click, and then choose New Record.

4. Type the name of the user property, such as Service Name, in the Name field.

5. Type the value of the user property, such as a business service name, in the Value field.

6. Repeat the process for every user property you want to define for this VBC.

**NOTE:** For the list of different property sets and their format, see "Examples of the Outgoing XML Format" on page 227 and "Examples of the Incoming XML Format" on page 231.
XML Gateway Service

The XML Gateway business service communicates between Siebel Business Applications and external data sources using XML as the data format. For information on XML format, see "Examples of the Outgoing XML Format" on page 227 and "Examples of the Incoming XML Format" on page 231. The XML Gateway business service can be configured to use one of the following transports:

- EAI MQSeries Server Transport
- EAI HTTP Transport
- EAI MSMQ Transport

You can configure the XML Gateway by specifying the transport protocol and the transport parameters you use in the Service Parameters User Property of the VBC, as shown in Table 40. When using the XML Gateway, specify the following user properties for your VBC.

Table 40. User Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Name</td>
<td>XML Gateway</td>
</tr>
<tr>
<td>Service Parameters</td>
<td>variable1 name=variable1 value; variable2 name=variable2 value;...</td>
</tr>
<tr>
<td>Remote Source</td>
<td>External Data Source</td>
</tr>
<tr>
<td>VBC Compatibility Mode</td>
<td>Siebel xxx, where xxx can be any Siebel release number.</td>
</tr>
</tbody>
</table>

**NOTE:** You can concatenate multiple name-value pairs using a semicolon (;), but do not use any spaces between the name, the equal sign, the value, and the semicolon.

For example, if you want to specify the EAI HTTP Transport, then you can use something like the following:

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

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

**To implement a VBC with MQSeries**

1. Call the EAI Business Integration Manager (Server Request) business service.
2. Define another service parameter for the name of a workflow to run, with the following user properties on the VBC:
   - **Service Name.** XML Gateway.
   - **Service Parameters.** Transport=EAI Business Integration Manager (Server Request);ProcessName=EAITEST.
Define a workflow, EAITEMST, to call the EAI MQSeries Server Transport with the SendReceive method.

Define a new process property, <Value>, on the workflow, and use it as an output argument on the EAI MQSeries Server Transport step in the workflow.

**XML Gateway Methods**

The XML Gateway provides the methods presented in Table 41.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td>Initializes the XML Gateway business service for every business component.</td>
</tr>
<tr>
<td>Delete</td>
<td>Deletes a given record in the remote data source.</td>
</tr>
<tr>
<td>Insert</td>
<td>Inserts a record into a remote data source.</td>
</tr>
<tr>
<td>PreInsert</td>
<td>Performs an operation that tests for the existence of the given business component. Only default values are returned from the external application.</td>
</tr>
<tr>
<td>Query</td>
<td>Queries the given business component from the given data source.</td>
</tr>
<tr>
<td>Update</td>
<td>Updates a record in the remote data source.</td>
</tr>
</tbody>
</table>

**XML Gateway Method Arguments**

The XML Gateway init, delete, insert, preInsert, query, and update methods take the arguments presented in Table 42.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote Source</td>
<td>The VBC Remote Source user property. The remote source from which the service is to retrieve data for the business component. This must be a valid connect string. When configuring the repository business component on top of the specialized business component class CSSBCVExten, you can define a user property Remote Source to allow the Transport Services to determine the remote destination and any connect information. If this user property is defined, then it is passed to every request as the remote-source tag.</td>
</tr>
<tr>
<td>Business Component Id</td>
<td>Unique key for the given business component.</td>
</tr>
</tbody>
</table>
### About Handling White Space

White space is handled by the XML parser while processing the request from the XML Gateway business service. When the white space is part of the XML syntax, it must be discarded by the XML parser and not preserved (passed on to the processing application). If the white space is in any other location (such as in element content within a document), then it must be preserved according to the XML specification, because it might have some meaning.

For example:

```xml
<mydata>
    <mytag>stuff</mytag>
</mydata>
```

Or this example:

```xml
<mydata><mytag>stuff</mytag></mydata>
```

are different to an XML parser.

To preserve white space, use the `xml:space` attribute with the value `preserve`, for example:

```xml
<mydata xml:space="preserve">
    <mytag>stuff and more stuff</mytag>
</mydata>
```

The value of `xml:space` applies to all children of the element containing the attribute unless overridden by one of the children.

For more information on white space handling and the `xml:space` attribute, see Microsoft Developer Network (http://msdn.microsoft.com).

---

**Table 42. XML Gateway Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Component Name</td>
<td>Name of the business component or its equivalent, such as a table name.</td>
</tr>
<tr>
<td>Parameters</td>
<td>The VBC Service Parameters user property. A set of string parameters required for initializing the XML Gateway.</td>
</tr>
</tbody>
</table>
Examples of the Outgoing XML Format

Examples of the XML documents generated and sent by the XML Gateway to the external system are presented in Table 43 on page 227. These examples are based on the Siebel eScript example in “Custom Business Service Examples” on page 250. 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 in this chapter have extraneous carriage returns and line feeds for ease of reading. Delete all the carriage returns and line feeds before using any of the examples.

<table>
<thead>
<tr>
<th>Method</th>
<th>Format of the XML Stream</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete Request</td>
<td><code>&lt;siebel-xmlext-delete-req&gt;</code>&lt;br&gt;<code>&lt;buscomp id=&quot;1&quot;&gt;Contact&lt;/buscomp&gt;</code>&lt;br&gt;<code>&lt;remote-source&gt;http://throth/servlet/VBCContacts&lt;/remote-source&gt;</code>&lt;br&gt;<code>&lt;row&gt;</code>&lt;br&gt;<code>&lt;value field=&quot;AccountId&quot;&gt;146&lt;/value&gt;</code>&lt;br&gt;<code>&lt;value field=&quot;Name&quot;&gt;Max Adams&lt;/value&gt;</code>&lt;br&gt;<code>&lt;value field=&quot;Phone&quot;&gt;(408)234-1029&lt;/value&gt;</code>&lt;br&gt;<code>&lt;value field=&quot;Location&quot;&gt;San Jose&lt;/value&gt;</code>&lt;br&gt;<code>&lt;value field=&quot;AccessId&quot;&gt;146&lt;/value&gt;</code>&lt;br&gt;<code>&lt;/row&gt;</code>&lt;br&gt;<code>&lt;/siebel-xmlext-delete-req&gt;</code></td>
<td>siebel-xmlext-delete-req&lt;br&gt;This tag requests removal of a single record in the remote system.</td>
</tr>
<tr>
<td>Init Request</td>
<td><code>&lt;siebel-xmlext-fields-req&gt;</code>&lt;br&gt;<code>&lt;buscomp id=&quot;1&quot;&gt;Contact&lt;/buscomp&gt;</code>&lt;br&gt;<code>&lt;remote-source&gt;http://throth/servlet/VBCContacts&lt;/remote-source&gt;</code>&lt;br&gt;<code>&lt;/siebel-xmlext-fields-req&gt;</code></td>
<td>siebel-xmlext-fields-req&lt;br&gt;This tag fetches the list of fields supported by this instance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>buscomp Id&lt;br&gt;The business component ID.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>remote-source&lt;br&gt;The remote source from which the service is to retrieve data for the business component.</td>
</tr>
</tbody>
</table>
### Table 43. Outgoing XML Tags and Descriptions

<table>
<thead>
<tr>
<th>Method</th>
<th>Format of the XML Stream</th>
<th>Description</th>
</tr>
</thead>
</table>
| Insert Request     | `<siebel-xmlext-insert-req>` `<buscomp id="1">Contact</buscomp>` `<remote-source>http://throth/servlet/VBCContacts</remote-source>` `<row>` `<value field="AccountId">1-6</value>` `<value field="Name">Max Adams</value>` `<value field="Phone">(398)765-1290</value>` `<value field="Location">Troy</value>` `<value field="AccessId"></value>` `</row>` `<siebel-xmlext-insert-req>` | siebel-xmlext-insert-req  
This tag requests the commit of a new record in the remote system.  
The insert-req XML stream contains values for fields entered through the business component. |
| PreInsert Request  | `<siebel-xmlext-preinsert-req>` `<buscomp id="1">Contact</buscomp>` `<remote-source>http://throth/servlet/VBCContacts</remote-source>` `<siebel-xmlext-preinsert-req>` | siebel-xmlext-preinsert-req  
This tag allows the connector to provide default values. This operation is called when a new row is created, but before any values are entered through the business component interface. |
### Table 43. Outgoing XML Tags and Descriptions

<table>
<thead>
<tr>
<th>Method</th>
<th>Format of the XML Stream</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Request</td>
<td><code>&lt;siebel-xmlext-query-req&gt;</code>&lt;br&gt;<code>&lt;buscomp id=&quot;1&quot;&gt;Contact&lt;/buscomp&gt;</code>&lt;br&gt;<code>&lt;remote-source&gt;http://throth/servlet/VBCContacts&lt;/remote-source&gt;</code>&lt;br&gt;<code>&lt;max-rows&gt;6&lt;/max-rows&gt;</code>&lt;br&gt;<code>&lt;search-string&gt;=[(Phone] IS NOT NULL) AND ([AccountId] = &quot;1-6&quot;)&lt;/search-string&gt;</code>&lt;br&gt;<code>&lt;search-spec&gt;</code>&lt;br&gt;<code>&lt;node node-type=&quot;Binary Operator&quot;&gt;AND&lt;/node&gt;</code>&lt;br&gt;<code>&lt;node node-type=&quot;Unary Operator&quot;&gt;IS NOT NULL&lt;/node&gt;</code>&lt;br&gt;<code>&lt;node node-type=&quot;Identifier&quot;&gt;Phone&lt;/node&gt;</code>&lt;br&gt;<code>&lt;/node&gt;</code>&lt;br&gt;<code>&lt;node node-type=&quot;Binary Operator&quot;&gt;=&lt;/node&gt;</code>&lt;br&gt;<code>&lt;node node-type=&quot;Identifier&quot;&gt;AccountId&lt;/node&gt;</code>&lt;br&gt;<code>&lt;/node&gt;</code>&lt;br&gt;<code>&lt;node value-type=&quot;TEXT&quot; node-type=&quot;Constant&quot;&gt;1-6&lt;/node&gt;</code>&lt;br&gt;<code>&lt;/node&gt;</code>&lt;br&gt;<code>&lt;/search-spec&gt;</code>&lt;br&gt;<code>&lt;sort-spec&gt;</code>&lt;br&gt;<code>&lt;sort field=&quot;Location&quot;&gt;ASCENDING&lt;/sort&gt;</code>&lt;br&gt;<code>&lt;sort field=&quot;Name&quot;&gt;DESCENDING&lt;/sort&gt;</code>&lt;br&gt;<code>&lt;/sort-spec&gt;</code>&lt;br&gt;<code>&lt;/Siebel-xmlext-query-req&gt;</code></td>
<td><strong>siebel-xmlext-query-req</strong>&lt;br&gt;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.&lt;br&gt;<strong>max-rows</strong>&lt;br&gt;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.&lt;br&gt;<strong>search-string</strong>&lt;br&gt;The search specification used to query and filter the information.&lt;br&gt;<strong>search-spec</strong>&lt;br&gt;Hierarchical representation of the search-string. For information, see “Search-Spec Node-Type Values” on page 230.&lt;br&gt;<strong>sort-spec</strong>&lt;br&gt;List of sort fields and sort order.&lt;br&gt;<strong>NOTE:</strong> In some cases you might retrieve external data for display in a child list applet, using a link to a parent business component. If the parent business component field on which the link is based is empty, then the query request is sent without a search-spec tag, but instead with the following tag:&lt;br&gt;<code>&lt;match field=&quot;Child BC Fieldname&quot;/&gt;</code></td>
</tr>
</tbody>
</table>
Search-Spec Node-Type Values

The search-string is in the Siebel query language format. The search-string is parsed by the Siebel query object and then turned into the hierarchical search-spec. Table 44 shows the different search-spec node-types and their values.

Table 44. Search-Spec Node-Types

<table>
<thead>
<tr>
<th>Node-Type</th>
<th>PropertySet and XML Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>Example: <code>&lt;node node-type = &quot;Constant&quot; value-type=&quot;NUMBER&quot;&gt;1000&lt;/node&gt;</code> The valid value-types are TEXT, NUMBER, DATETIME, UTCDATETIME, DATE, and TIME.</td>
</tr>
<tr>
<td>Identifier</td>
<td>Example: <code>&lt;node node-type=&quot;Identifier&quot;&gt;Name&lt;/node&gt;</code> The value Name is a valid business component field name.</td>
</tr>
<tr>
<td>Unary Operator</td>
<td>Example: <code>&lt;node node-type=&quot;Unary Operator&quot;&gt;NOT&lt;/node&gt;</code> The valid values are NOT, EXISTS, IS NULL, IS NOT NULL.</td>
</tr>
<tr>
<td>Binary Operator</td>
<td>Example: <code>&lt;node node-type=&quot;Binary Operator&quot;&gt;AND&lt;/node&gt;</code> The valid values are LIKE, NOT LIKE, SOUNDSLIKE, =, &lt;&gt;, &lt;=, &lt;, &gt;=, &gt;, AND, OR, +, -, *, /, ^.</td>
</tr>
</tbody>
</table>
Examples of the Incoming XML Format

Table 45 contains examples of XML documents that are sent from an external system to the XML Gateway in response to a request. These examples are based on the Siebel eScript example in “Custom Business Service Examples” on page 250. See Appendix C, “DTDs for XML Gateway Business Service,” for examples of the DTDs that correspond to each of these methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Format of the XML Stream</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete Return</td>
<td><code>&lt;siebel-xmlext-delete-ret /&gt;</code></td>
<td>siebel-xmlext-delete-ret. Only the XML stream tag is returned.</td>
</tr>
<tr>
<td>Error</td>
<td><code>&lt;siebel-xmlext-status&gt;</code></td>
<td>Format of the XML stream expected by the Siebel application in case of an error in the external application. If the error is specific to a field, then the field name must be specified. The tags for this XML stream, and the entire XML stream, are optional:</td>
</tr>
<tr>
<td></td>
<td><code>&lt;status-code&gt;</code></td>
<td>This tag is used to check the status returned by the external system.</td>
</tr>
<tr>
<td></td>
<td><code>&lt;error-field&gt;</code></td>
<td>This tag overrides the return value.</td>
</tr>
<tr>
<td></td>
<td><code>&lt;error-text&gt;</code></td>
<td>This tag specifies textual representation of the error, if it is available. This tag appears in addition to the standard error message. For example, if the Siebel application attempts to update a record in the external system with a NULL Name, and this is not allowed in the external system, then the error text is set to: “Name must not be empty.”</td>
</tr>
</tbody>
</table>
### Table 45. Incoming XML Tags and Descriptions

<table>
<thead>
<tr>
<th>Method</th>
<th>Format of the XML Stream</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init Return</td>
<td><code>&lt;siebel-xmlext-fields-ret&gt;</code></td>
<td>siebel-xmlext-fields-ret</td>
</tr>
<tr>
<td></td>
<td>```</td>
<td>The fields-ret XML stream return contains the list of VBC fields supported by the external application for this instance. The following field names are reserved by the Siebel application, and must not appear in this list:</td>
</tr>
<tr>
<td></td>
<td>```&lt;support field=&quot;AccountId&quot;/&gt;</td>
<td>■ Id</td>
</tr>
<tr>
<td></td>
<td>```&lt;support field=&quot;Name&quot;/&gt;</td>
<td>■ Created</td>
</tr>
<tr>
<td></td>
<td>```&lt;support field=&quot;Phone&quot;/&gt;</td>
<td>■ Created By</td>
</tr>
<tr>
<td></td>
<td>```&lt;support field=&quot;Location&quot;/&gt;</td>
<td>■ Updated</td>
</tr>
<tr>
<td></td>
<td>```&lt;support field=&quot;AccessId&quot;/&gt;</td>
<td>■ Updated By</td>
</tr>
<tr>
<td></td>
<td>```&lt;siebel-xmlext-fields-ret&gt;</td>
<td></td>
</tr>
<tr>
<td>Insert Return</td>
<td><code>&lt;siebel-xmlext-insert-ret&gt;</code></td>
<td>siebel-xmlext-insert-ret</td>
</tr>
<tr>
<td></td>
<td>```&lt;row&gt;</td>
<td>If the remote system has inserted records, then they can be returned to be reflected in the business component in an insert-ret XML stream in the row tag format as the insert-ret stream.</td>
</tr>
<tr>
<td></td>
<td>```&lt;value field=&quot;AccountId&quot;&gt;1-6&lt;/value&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>```&lt;value field=&quot;Name&quot;&gt;Max Adams&lt;/value&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>```&lt;value field=&quot;Phone&quot;&gt;(398)765-1290&lt;/value&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>```&lt;value field=&quot;Location&quot;&gt;Troy&lt;/value&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>```&lt;value field=&quot;AccessId&quot;&gt;146&lt;/value&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>```&lt;row&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>```&lt;siebel-xmlext-insert-ret&gt;</td>
<td></td>
</tr>
<tr>
<td>PreInsert Return</td>
<td><code>&lt;siebel-xmlext-preinsert-ret&gt;</code></td>
<td>siebel-xmlext-preinsert-ret</td>
</tr>
<tr>
<td></td>
<td>```&lt;row&gt;</td>
<td>Returns default values for each field, if there is any default value.</td>
</tr>
<tr>
<td></td>
<td>```&lt;value field=&quot;Location&quot;&gt;San Jose&lt;/value&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>```&lt;siebel-xmlext-preinsert-ret&gt;</td>
<td></td>
</tr>
</tbody>
</table>
### Table 45. Incoming XML Tags and Descriptions

<table>
<thead>
<tr>
<th>Method</th>
<th>Format of the XML Stream</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Return</td>
<td><em>&lt;siebel-xmlext-query-ret&gt;</em></td>
<td><strong>siebel-xmlext-query-ret</strong>&lt;br&gt;The query-ret XML stream contains the result set that matches the criteria of the query.</td>
</tr>
<tr>
<td></td>
<td><em>&lt;row&gt;</em></td>
<td><strong>row</strong>&lt;br&gt;This tag indicates the number of rows returned by the query. Each row must contain one or more value tags. The attributes that appear in row tags must be able to uniquely identify the rows. If there is a unique key in the remote data source, then it appears in the result set. If not, a unique key is generated. It is necessary to identify specific rows for DML operations.</td>
</tr>
<tr>
<td></td>
<td><em>&lt;value field=&quot;AccountId&quot;&gt;1-6&lt;/value&gt;</em></td>
<td><strong>value</strong>&lt;br&gt;This tag specifies the field and value pairs and must be the same for each row in the set.</td>
</tr>
<tr>
<td></td>
<td><em>&lt;value field=&quot;Name&quot;&gt;Sara Chen&lt;/value&gt;</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>&lt;value field=&quot;Phone&quot;&gt;(415)298-7890&lt;/value&gt;</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>&lt;value field=&quot;Location&quot;&gt;San Francisco&lt;/value&gt;</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>&lt;value field=&quot;AccessId&quot;&gt;128&lt;/value&gt;</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>/row&gt;</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>&lt;row&gt;</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>&lt;value field=&quot;AccountId&quot;&gt;1-6&lt;/value&gt;</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>&lt;value field=&quot;Name&quot;&gt;Eric Brown&lt;/value&gt;</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>&lt;value field=&quot;Phone&quot;&gt;(650)123-1000&lt;/value&gt;</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>&lt;value field=&quot;Location&quot;&gt;Palo Alto&lt;/value&gt;</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>&lt;value field=&quot;AccessId&quot;&gt;129&lt;/value&gt;</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>/row&gt;</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>/siebel-xmlext-query-ret</em></td>
<td></td>
</tr>
</tbody>
</table>
External Application Setup

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

Custom Business Service Methods

Your business service must implement the Init and Query methods as described in this topic. The Delete, PreInsert, Insert, and Update methods are optional, and depend on the functionality required by the VBC.

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

These methods pass property sets between the VBC and the business service. VBC methods take property sets as arguments. Each method takes two property sets: an Inputs property set and an Outputs property set. The methods are called by the `CSSBCVExtern` class in response to requests from other objects that refer to, or are based on the VBC.

If VBCs are used, then custom business services are written to access external relational databases. However, it is recommended that you use external business components (EBCs) to access these databases instead of writing custom business services. For more information on EBCs, see Chapter 10, "External Business Components."

Table 45. Incoming XML Tags and Descriptions

<table>
<thead>
<tr>
<th>Method</th>
<th>Format of the XML Stream</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update Return</td>
<td><code>&lt;siebel-xmlext-update-ret&gt;</code>&lt;br&gt;<code>&lt;row&gt;</code>&lt;br&gt;<code>&lt;value field=&quot;Location&quot;&gt;San Jose&lt;/value&gt;</code>&lt;br&gt;<code>&lt;value field=&quot;Phone&quot;&gt;(408)234-1029&lt;/value&gt;</code>&lt;br&gt;<code>&lt;/row&gt;</code>&lt;br&gt;<code>&lt;/siebel-xmlext-update-ret&gt;</code></td>
<td>siebel-xmlext-update-ret&lt;br&gt;If the remote system updated fields, then the fields can be returned to be reflected in the business component in an update-ret XML stream in the row tag format as the update-ret stream.</td>
</tr>
</tbody>
</table>
Common Method Parameters

Table 46 shows the input parameters common to every method. Note that all these parameters are at the root property set.

Table 46. Common Input Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote Source</td>
<td>(Optional) Specifies the name of an external data source. This is the VBC’s Remote Source user property, if defined. For information, see Table 39 on page 222.</td>
</tr>
<tr>
<td>Business Component Name</td>
<td>Name of the active VBC.</td>
</tr>
<tr>
<td>Business Component Id</td>
<td>Internally generated unique value that represents the VBC.</td>
</tr>
<tr>
<td>Parameters</td>
<td>(Optional) The VBC’s Service Parameters user property, if defined. For information, see Table 39 on page 222. A set of parameters required by the business service.</td>
</tr>
<tr>
<td>VBC Compatibility Mode</td>
<td>(Optional) This is the VBC’s Compatibility Mode user property, if defined. For information, see Table 39 on page 222.</td>
</tr>
</tbody>
</table>

When a response has been received, the method packages the response from the external data source into the output’s property set.

Business Services Methods and Their Property Sets

The following examples display each method’s input and output property sets for a VBC Contact that displays simple contact information for a given account. These examples are based on the example in the “Custom Business Service Examples” on page 250.

The output property set of the Insert and Update methods for VBC does not affect the data in the business component, unlike the Query method, which uses the output property set to populate the business component. The output property set for Insert and Update is used to indicate that what fields or record has been changed.

**NOTE:** All the optional parameters have been omitted from these examples to simplify them.
Delete Method

The Delete method is called when a record is deleted. Figure 36 illustrates the property set for the Delete input.

The following is the XML representation of the property set shown in Figure 36:

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

<PropertySet>
  <Business_component_spcId="1">
    <Business_component_spcName="Contact">
      <PropertySet>
        <AccountId>1-6</AccountId>
        <Name>Max Adams</Name>
        <Phone>(408)234-1029</Phone>
        <Location>San Jose</Location>
        <AccessId>146</AccessId>
      </PropertySet>
    </Business_component_spcName>
  </Business_component_spcId>
</PropertySet>
```

Figure 36. Delete Input Property Set
Figure 37 illustrates the property set for the Delete output.

![Property Set](image)

**Figure 37. Delete Output Property Set**

The following is the XML representation of the property set shown in Figure 37:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<PropertySet EscapeNames="true"/>
```
Error Return Method

Figure 38 illustrates the property set for the Error Return, when an error is detected.

![Property Set Diagram]

Figure 38. Error Return Property Set

The following is the XML representation of the property set shown in Figure 38 on page 238:

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

<PropertySet>
    <Status Status="4"
    Error_spcField="Name"
    Error_spcText="Name must not be empty"/>
</PropertySet>
```

Init Method

The Init method is called when the VBC is first instantiated. It initializes the VBC. It expects to receive the list of fields supported by the external system.

**NOTE:** When a field is not initialized in the Init method of the VBC, the Update method is not fired when the field gets updated.
Figure 39 illustrates the property set for the Init input.

![Property Set](image)

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Component Id</td>
<td>1</td>
</tr>
<tr>
<td>Business Component Name</td>
<td>Contact</td>
</tr>
</tbody>
</table>

The following is the XML representation of the property set shown in Figure 39:

```xml
<?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 the Init output.

![Property Set](image)

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccountId</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>Phone</td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td></td>
</tr>
<tr>
<td>AccessId</td>
<td></td>
</tr>
</tbody>
</table>

The following is the XML representation of the property set shown in Figure 40:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet
  AccountId="" />
```
Insert Method

The Insert method is called when a New Record is committed. Figure 41 illustrates the property set for the Insert input.

The following is the XML representation of the property set shown in Figure 41:

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

Figure 41. Insert Input Property Set
Figure 42 illustrates the property set for the Insert output.

**NOTE:** The property set for the Insert output does not affect the data in the business component. The output property set for Insert is used to indicate what fields or records were changed.

The following is the XML representation of the property set shown in **Figure 42**:

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

Figure 42. Insert Output Property Set
### PreInsert Method

The PreInsert method is called when a New Record operation is performed. It supplies default values. Figure 43 illustrates the property set for the PreInsert input.

#### Figure 43. PreInsert Input Property Set

The following is the XML representation of the property set shown in Figure 43:

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

<PropertySet
    Business_spcComponent_spclid="1"
    Business_spcComponent_spcName="Contact"/>
```

---

Integration Platform Technologies: Siebel Enterprise Application Integration Version 8.1/8.2
Figure 44 illustrates the property set for the PreInsert output.

The following is the XML representation of the property set shown in Figure 44:

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

**Query Method**

The Query method is called when a search is performed. The Query method must be supported by every VBC. Each record that matches the query is represented as a property set. For example, if 5 records match the query, then there will be 5 child property sets. Each property set contains a list of field names; that is, field value pairs representing the values of each field for that particular record. Figure 45 on page 244 illustrates the property set for the Query input and is followed by its XML representation.
Figure 45. Query Input Property Set
The following is the XML representation of the property set shown in Figure 45 on page 244:

```xml
<?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_spclId="1"
  Business_spcComponent_spcName="Contact">
  <PropertySet AccountId="1-6" />
  <search-spec>
    <node node-type="Binary Operator">AND</node>
    <node node-type="Unary Operator">IS NOT NULL</node>
    <node node-type="Identifier">Phone</node>
  </node>
  <node node-type="Binary Operator">=</node>
  <node node-type="Identifier">AccountId</node>
  <node value-type="TEXT" node-type="Constant">1-6</node>
</search-spec>
<sort-spec>
  <sort field="Location">ASCENDING</sort>
  <sort field="Name">DESCENDING</sort>
</sort-spec>
</PropertySet>
```
Figure 46 illustrates the property set for the Query output.

```xml
<PropertySet>
  <PropertySet AccountId="1-6" />
  <PropertySet AccountId="16">
    <AccountName>Sara Chen</AccountName>
    <Phone>(415) 298-7890</Phone>
    <Location>San Francisco</Location>
    <AccessId>128</AccessId>
  </PropertySet>
  <PropertySet AccountId="1-6">
    <AccountName>Eric Brown</AccountName>
    <Phone>(650) 123-1000</Phone>
    <Location>Palo Alto</Location>
    <AccessId>129</AccessId>
  </PropertySet>
</PropertySet>
```

Figure 46. Query Output Property Set

The following is the XML representation of the property set shown in Figure 46:
Update Method
The Update method is called when a record is modified. Figure 47 illustrates the property set for the Update input.

Figure 47. Update Input Property Set

The following is the XML representation of the property set shown in Figure 47:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet
    Business_spccomponent_spclid="1"
    Business_spccomponent_spccname="Contact">
    <Property Name="Phone" Value="(650)123-1000" />
    <Property Name="Location" Value="Palo Alto" />
    <Property Name="AccessId" Value="129" />
</PropertySet>
```
<PropertySet>
  <Field_spcName="AccountId">
    Changed="false"
    Field_spcValue="1-6"/
  </PropertySet>

<PropertySet>
  <Field_spcName="Name">
    Changed="false"
    Field_spcValue="MaxAdams"/
  </PropertySet>

<PropertySet>
  <Field_spcName="Phone">
    Changed="true"
    Field_spcValue="(408)234-1029"/
  </PropertySet>

<PropertySet>
  <Field_spcName="Location">
    Changed="true"
    Field_spcValue="SanJose"/
  </PropertySet>

<PropertySet>
  <Field_spcName="AccessId">
    Changed="false"
    Field_spcValue="146" /
  </PropertySet>
</PropertySet>
Figure 48 illustrates the property set for the Update output.

**NOTE:** The property set for Update output does not affect the data in the business component. The output property set for Update is used to indicate what fields or records were changed.

The following is the XML representation of the property set shown in Figure 48:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet>
  <PropertySet>
    <Property>>Phone="(408) 234-1029"
    <Property>>Location="San Jose" />
  </PropertySet>
</PropertySet>
```
Custom Business Service Examples

These examples show the implementation of a business service for a VBC in both Siebel eScript and Siebel VB:

- “Siebel eScript Business Service Example for a VBC” on page 250
- “Siebel VB Business Service Example for a VBC” on page 257

Siebel eScript Business Service Example for a VBC

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

**CAUTION:** Do not use Siebel CRM system fields, such as Id, as output properties. Problematic application behavior might result.

```javascript
function Service_PreInvokeMethod (MethodName, Inputs, Outputs) {
    if (MethodName == "Init") {
        return(Init(Inputs, Outputs));
    } else if (MethodName == "Query") {
        return(Query(Inputs, Outputs));
    } else if (MethodName == "PreInsert") {
        return(PreInsert(Inputs, Outputs));
    } else if (MethodName == "Insert") {
        return(Insert(Inputs, Outputs));
    } else if (MethodName == "Update") {
        return(Update(Inputs, Outputs));
    } else if (MethodName == "Delete") {
        return(Delete(Inputs, Outputs));
    } else {
        return (ContinueOperation);
    }
}

function Init (Inputs, Outputs) {
    // For debugging purposes...
    logPropSet(Inputs, "InitInputs.xml");
    Outputs.SetProperty("AccountId", "");
    Outputs.SetProperty("Name", "");
    Outputs.SetProperty("Phone", "");
    Outputs.SetProperty("AccessId", "");
    Outputs.SetProperty("Location", "");
    // For debugging purposes...
```
logPropSet(Outputs, "InitOutputs.xml");
return (CancelOperation);
}

function Query(Inputs, Outputs) {
  // For debugging purposes...
  logPropSet(Inputs, "QueryInputs.xml");
  var selectStmt = "select * from Contacts ";
  var whereClause = "";
  var orderByClause = "";
  // You have the following properties if you want to use them
  // Inputs.GetProperty("Business Component Name")
  // Inputs.GetProperty("Business Component Id")
  // Inputs.GetProperty("Remote Source")
  // If you configured Maximum Cursor Size at the buscomp,
  // get max-rows property
  var maxRows = Inputs.GetProperty("max-rows");
  // get search-string
  var searchString = Inputs.GetProperty("search-string");
  if (searchString != "")
  {
    // convert the search-string into a where clause
    searchString = stringReplace(searchString, '*', '%");
    searchString = stringReplace(searchString, '[', '");
    searchString = stringReplace(searchString, ']', '");
    searchString = stringReplace(searchString, '~', '");
    searchString = stringReplace(searchString, '"', '");
    whereClause = " where ";
    whereClause = whereClause + searchString;
  }
  // match, search-spec, sort-spec
  var childCount = Inputs.GetChildCount();
  var child, sortProp;
  for (var i = 0; i < childCount; i++)
  {
    child = Inputs.GetChild(i);
    if (child.GetType() == "")
    {
      // Use this child property set if you want to use the old match field list.
      // We are not using this in this example. We'll use search-string instead.
    }
    else if (child.GetType() == "search-spec")
    {
      // Use this child property set if you want to use the hierarchical
      // representation of the search-string.
      // We are not using this in this example. We'll use search-string instead.
    }
    else if (child.GetType() == "sort-spec")
    {
      // This child property set has the sort spec. We'll use this in this example
      orderByClause = " order by ";
      var sortFieldCount = child.GetChildCount();
      for (var j = 0; j < sortFieldCount; j++)


```csharp
{
    // 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 rs = getConnection();
rs.Open(selectStmt, conn);
// We will return no more than maxRows of records.
var count = rs.RecordCount;
if (maxRows != "")
    if (count > maxRows)
        count = maxRows
// Iterate through the record set and add them to the Outputs PropertySet.
var fcount, fields, row;
for (i = 0; i < count; i++)
{
    row = TheApplication().NewPropertySet();
    fields = rs.Fields();
    fcount = fields.Count;
    for (j = 0; j < fcount; j++)
    {
        var fieldValue = fields.Item(j).Value;
        if (fieldValue == null)
            row.SetProperty(fields.Item(j).Name, "");
        else
            row.SetProperty(fields.Item(j).Name, fieldValue);
    }
    Outputs.AddChild(row);
    rs.MoveNext();
}
// For debugging purposes...
logPropSet(Outputs, "QueryOutputs.xml");
// clean up
child = null;
sortProp = null;
row = null;
rs.Close();
rs = null;
conn.Close();
conn = null;
return (CancelOperation);
}
```
function PreInsert (Inputs, Outputs) {
    // For debugging purposes...
    logPropSet(Inputs, "PreInsertInputs.xml");
    var defaults = TheApplication().NewPropertySet();
    defaults.SetProperty("Location", "KO");
    Outputs.AddChild(defaults);
    // For debugging purposes...
    logPropSet(Outputs, "PreInsertOutputs.xml");
    // Cleanup
    defaults = null;
    return (CancelOperation);
}

function Insert (Inputs, Outputs) {
    // For debugging purposes...
    logPropSet(Inputs, "InsertInputs.xml");
    var fieldList = "";
    var valueList = "";
    // Inputs must have only 1 child property set.
    var child = Inputs.GetChild(0);
    var fieldName = child.GetFirstProperty();
    var fieldValue;
    while (fieldName != "")
    {
        fieldValue = child.GetProperty(fieldName);
        if (fieldValue != "")
        {
            if (fieldList != "")
            {
                fieldList += ", ";
                valueList += ", ";
            }
            fieldList += fieldName;
            valueList += "'" + fieldValue + "'";
        }
        fieldName = child.GetNextProperty();
    }
    // The insert statement is...
    var insertStmt = "insert into Contacts (" + fieldList + ") values (" + valueList + ")";
    // Now, inserting into the data source...
    var conn = getConnection();
    conn.Execute(insertStmt);
    // In this example, we must query back the record just inserted to get
    // the value of its primary key. We made this primary key part of the buscomp
    // to make update and delete easy. The primary key is "AccessId".
    var selectStmt = "select * from Contacts where ";
    var whereClause = "";
    child = Inputs.GetChild(0)
    fieldName = child.GetFirstProperty();
    while (fieldName != "")
    {
        fieldValue = child.GetProperty(fieldName);
        if (fieldName != "AccessId")
        {
            if (fieldValue != "AccessId")
            {
if (whereClause != "")
    whereClause += ' and ';
if (fieldValue == '')
    whereClause += fieldName + ' is null';
else
    whereClause += fieldName + '=' + fieldValue + ' ';
}
fieldName = child.GetNextProperty();
// The select statement is...
selectStmt += whereClause;
// Now, let's select the new record back
var rs = getRecordset();
rs.Open(selectStmt, conn);
// We're expecting only one row back in this example.
var fcount, fields, row, fieldValue;
row = TheApplication().NewPropertySet();
fields = rs.Fields();
fcount = fields.Count();
for (var j = 0; j < fcount; j++)
{
    fieldValue = fields.Item(j).Value();
    if (fieldValue == null)
        row.SetProperty(fields.Item(j).Name(), '');
    else
        row.SetProperty(fields.Item(j).Name(), fieldValue);
}
Outputs.AddChild(row);
// For debugging purpose...
logPropSet(Outputs, "InsertOutputs.xml");
// Cleanup
child = null;
row = null;
rs.Close();
rs = null;
conn.Close();
conn = null;
return (CancelOperation);
}

function Update (Inputs, Outputs) {
// For debugging purposes...
logPropSet(Inputs, "UpdateInputs.xml");
var child;
var childCount = Inputs.GetChildCount();
var fieldName, fieldValue;
var updateStmt = 'update Contacts set ';
var setClause = '';
var whereClause;
// Go through each child in Inputs and construct the necessary sql statements for update and query
for (var i = 0; i < childCount; i++)
{
    child = Inputs.GetChild(i);
    var childCondition = child.GetNextProperty();
    if (childCondition != '')
        setClause += childCondition + ' = ' + childValue + ' and ';
    else
        setClause += childCondition + ' = ' + childValue + ' ';
fieldName = child.GetProperty("Field Name");
fieldValue = child.GetProperty("Field Value");
// We only have to update changed fields.
if (child.GetProperty("Changed") == "true")
{
    if (setClause != ")
        setClause += ";
    if (fieldValue == ")
        setClause += fieldName + "=null";
    else
        setClause += fieldName + "='" + fieldValue + ");
}
if (fieldName == "AccessId")
    whereClause = " where AccessId = " + fieldValue;
// The update statement is...
updateStmt += setClause + whereClause;
// Now, updating the data source...
var conn = getConnection();
conn.Execute(updateStmt);
// How to construct the Outputs PropertySet can vary, but in this example
// We'll query back the updated record from the data source.
var selectStmt = "select * from Contacts" + whereClause;
// Now, let's select the updated record back
var rs = getRecordset();
rs.Open(selectStmt, conn);
// We expect only one row back in this example.
// In this example, we're returning all the fields and not just
// the updated fields. You can only return those updated
// fields with the new value in the Outputs property set.
var fcount, fields, row, fieldValue;
row = TheApplication().NewPropertySet();
fields = rs.Fields();
fcount = fields.Count();
for (var j = 0; j < fcount; j++)
{
    fieldValue = fields.Item(j).Value();
    if (fieldValue == null)
        row.SetProperty(fields.Item(j).Name(), ");
    else
        row.SetProperty(fields.Item(j).Name(), fieldValue);
}
Outputs.AddChild(row);
// For debugging purposes...
logPropSet(Outputs, "UpdateOutputs.xml");
// Cleanup
child = null;
row = null;
rs.Close();
rs = null;
conn.Close();
conn = null;
return (CancelOperation);
function Delete (Inputs, Outputs) {
    // For debugging purposes...
    logPropSet(Inputs, "DeleteInputs.xml");
    // Inputs must have only 1 child property set.
    var child = Inputs.GetChild(0);
    // In this example, we’re only using the AccessId
    // (it is the primary key in the Contacts db)
    // for the delete statement for simplicity.
    var deleteStmt = "delete from Contacts where AccessId = " + child.GetProperty("AccessId");
    // Now, delete the record from the data source.
    var conn = getConnection();
    conn.Execute(deleteStmt);
    // For debugging purposes...
    logPropSet(Outputs, "DeleteOutputs.xml");
    // Returning empty Outputs property set.
    // clean up
    conn.Close();
    conn = null;
    return {CancelOperation};
}

The following functions are helper functions:

function getConnection () {  
    // VBCContact is the ODBC data source name
    var connectionString = "DSN=VBCContact";
    var uid = "";
    var passwd = "";
    var conn = COMCreateObject("ADODB.Connection");
    conn.Mode = 3;
    conn.CursorLocation = 3;
    conn.Open(connectionString, uid, passwd);
    return conn;
}

function getRecordset() {  
    var rs = COMCreateObject("ADODB.Recordset");
    return rs;
}

function logPropSet(inputPS, fileName) {  
    // Use EAI XML Write to File business service to write
    // inputPS property set to fileName file in c:\temp directory.
    var fileSvc = TheApplication().GetService("EAI XML Write to File");
    var outPS = TheApplication().NewPropertySet();
    var fileLoc = "c:\temp\" + fileName;
    var tmpProp = inputPS.Copy();
    tmpProp.SetParentProperty("FileName", fileLoc);
    fileSvc.InvokeMethod("WritePropSet", tmpProp, outPS);
    // clean up
    outPS = null;
    fileSvc = null;
    tmpProp = null;
}
function stringReplace (string, from, to) {
    // Replaces from with to in string
    var stringLength = string.length;
    var fromLength = from.length;
    if ((stringLength == 0) || (fromLength == 0))
        return string;
    var fromIndex = string.indexOf(from);
    if (fromIndex < 0)
        return string;
    var newString = string.substring(0, fromIndex) + to;
    if ((fromIndex + fromLength) < stringLength)
        newString += stringReplace(string.substring(fromIndex+fromLength, stringLength), from, to);
    return newString;
}

Siebel VB Business Service Example for a VBC

The following is an example of Siebel VB implementation of a business service for a VBC. The fields configured for this simple VBC are AccountId, Name, Phone, and Location.

CAUTION: Do not use Siebel CRM system fields, such as Id, as output properties. Undesired application behavior might result.

```vbnet
 Option Explicit
 Declare Function stringReplace(ByVal mystr As String, ByVal fromchar As String, ByVal tochar As String) As String
 Declare Function getData(ByVal execSQL As String, ByVal Results As PropertySet) As Integer
 Function getData(ByVal execSQL As String, ByVal Results As PropertySet) As Integer
     Dim sSrv As String, sDbn As String
     Dim sUsr As String, sPsw As String
     Dim oCon As Object, oRec As Object
     Dim Row As PropertySet
     Dim FileName, TextToSave
     ' *** SQL Server connectivity parameters
     sSrv = "v817.siebel.com" '*** Oracle tns
     sUsr = "system" '*** SQL Server: a user's login ID
     sPsw = "manager" '*** SQL Server: a user's password
     ' *** Create SQL Server ADODB connection dynamically
     Set oCon = CreateObject("ADODB.Connection")
     oCon.Open "Provider=MSDAORA;" & _
         "Data Source=" & sSrv & ";" & _
         "User ID=" & sUsr & ";" & _
         "Password=" & sPsw & ";" & _
     ' *** Perform SQL query
     Set oRec = oCon.Execute(execSQL)
     ' *** Process SQL query result and save into file
     While Not oRec.EOF
         Set Row=TheApplication.NewPropertySet()
         Row.SetProperty "AccountId", oRec.Fields.Item("AccountId").Value
         Row.SetProperty "Name", oRec.Fields.Item("Name").Value
     W```
Row.SetProperty "Location", oRec.Fields.Item("Location").Value
Row.SetProperty "Phone", oRec.Fields.Item("Phone").Value
Results.AddChild Row
Set Row = Nothing
oRec.MoveNext
Wend
' *** Object cleanup
Set oRec = Nothing
Set oCon = Nothing
getData = 0
End Function

Sub Init(Inputs As PropertySet, Outputs As PropertySet)
Outputs.SetProperty "AccountId", ""
Outputs.SetProperty "Name", ""
Outputs.SetProperty "Phone", ""
Outputs.SetProperty "Location", ""
End Sub

Sub Query(Inputs As PropertySet, Outputs As PropertySet)
Dim sselectStmt As String
Dim swhereClause As String
Dim sorderbyClause As String
Dim ssearchstring As String
Dim child As PropertySet
Dim sortProp As PropertySet
Dim childCount As Integer
Dim i As Integer
Dim ret As Integer
Dim FileName, TextToSave
sselectStmt = "select * from siebel.Contact2"
swhereClause = "where"
sorderbyClause = "order by"
ssearchstring = Inputs.GetProperty("search-string")
If Len(ssearchstring) > 0 Then
    ssearchstring = stringReplace(ssearchString, "*", "%")
    ssearchstring = stringReplace(ssearchString, ";", "")
    ssearchstring = stringReplace(ssearchString, ";", "")
    ssearchstring = stringReplace(ssearchString, ";", "")
    ssearchstring = stringReplace(ssearchString, "'", "")
    ssearchstring = stringReplace(ssearchString, chr$(34), "")
    sselectStmt = sselectStmt & swhereClause & ssearchstring
End If
" Write select statement to this file
FileName = "C:\Test.txt"
TextToSave = "select is " & sselectStmt
Open FileName For Append As #1
Print #1, TextToSave
Close #1
ret = getData(sselectStmt, Outputs)
End Sub

Function stringReplace(mystri As String, fromchar As String, tochar As String) As String
' Replace all occurrences of fromchar in mystri with tochar
Dim i As Long
If Len(mystr) = 0 Or Len(fromchar) = 0 Then
    stringReplace = mystr
Else
    i = InStr(1, mystr, fromchar)
    Do While i > 0
        mystr = Left(mystr, i - 1) & tochar & Mid(mystr, i + Len(fromchar))
        i = i + Len(fromchar)
    Loop
    stringReplace = mystr
End If
End Function

Function Service_PreInvokeMethod (MethodName As String, Inputs As PropertySet, Outputs As PropertySet) As Integer
    Service_PreInvokeMethod = ContinueOperation
    If MethodName = "Init" Then
        Service_PreInvokeMethod = CancelOperation
        Init Inputs, Outputs
        Exit Function
    End If
    If MethodName = "Query" Then
        Service_PreInvokeMethod = CancelOperation
        Query Inputs, Outputs
        Exit Function
    End If
End Function
Siebel EAI supports file attachments for exchanging business documents such as sales literature, activity attachments, and product defect attachments with another Siebel instance or an external system such as Oracle Applications. It includes the following topics:

- About File Attachments on page 261
- Exchanging Attachments with External Applications on page 261
- Using MIME Messages to Exchange Attachments on page 262
- About the EAI MIME Hierarchy Converter on page 269
- About the EAI MIME Doc Converter on page 270
- Using Inline XML to Exchange Attachments on page 274

### About File Attachments

For example, if you are exchanging service requests with another application or partner, then you can include attachments such as screen captures, email, log files, and contract agreements that are associated with the service request in the information being exchanged. Siebel EAI support for file attachments allows comprehensive integration.

To use file attachments you first must create integration objects. For information, see Chapter 2, “Integration Objects,” and Chapter 3, “Creating and Maintaining Integration Objects.”

Siebel EAI offers the choice of integrating file attachments using MIME (the industry standard for exchanging multipart messages), or including the attachment within the body of the XML document, referred to as an inline XML attachment. Consider using inline XML attachments when integrating two instances of Siebel Business Applications using file attachments.

### Exchanging Attachments with External Applications

Siebel EAI supports bidirectional attachment exchange with external applications using the following two message types:

- **MIME (Multipurpose Internet Mail Extensions).** MIME is the industry standard for exchanging multipart messages. The first part of the MIME message is an XML document representing the business object being exchanged and attachments to the object are included as separate parts of the multipart message. MIME is the recommended choice for integrating Siebel Business Applications with other applications. For more information, see “Using MIME Messages to Exchange Attachments” on page 262.
Inline XML attachments (Inline Extensible Markup Language). With inline XML attachments, the entire business object you are exchanging, including any attachments, is sent as a single XML file. Consider using inline XML attachments when integrating two instances of Siebel Business Applications using file attachments. For more information, see “Using Inline XML to Exchange Attachments” on page 274.

Using MIME Messages to Exchange Attachments

To send or receive file attachments using MIME messages, Siebel EAI uses the MIME Hierarchy Converter and MIME Doc Converter.

You must perform the following procedures to use MIME to exchange attachments between Siebel Business Applications and another external system:

- Create an attachment integration object using the EAI Siebel Wizard business service.
  
  For information, see “Creating an Attachment Integration Object” on page 262.

- Create an inbound or outbound workflow.
  
  For information, see “Creating Workflow Examples” on page 264.

- Test your workflow using the Workflow Simulator.
  
  For information, see Business Processes and Rules: Siebel Enterprise Application Integration.

Creating an Attachment Integration Object

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

To create a new attachment integration object

1. In Siebel Tools, create a new project and lock it, or lock an existing project in which you want to create your integration object.

2. From the File menu, choose New Object to display the New Object Wizards dialog box.

3. Select the EAI tab, and then double-click Integration Object.
   
   The Integration Object Builder wizard appears.
4 Follow the procedure in "Creating Integration Objects Using the EAI Siebel Wizard Business Service" on page 45 to create the new integration object, for example SourceObject Attachment.

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

5 In the Object Explorer, select Integration Object, and then select your new integration object in the Object List Editor.

6 In the Object Explorer, expand the Integration Object tree to show the Integration Component object.

7 Select the SourceObject Attachment integration component, and set its External Sequence and XML Sequence properties so that they are greater than those of the other integration components (that is, last in sequence), if not already set.

If they are not last, the situation can arise where the attachment is processed successfully (and the file system is physically updated). Then a subsequent integration component causes a failure (for example, an attempt to insert to the database causes a duplicate error). In this case, the database transaction is rolled back, but the file system is not restored.

8 With the SourceObject Attachment integration component selected, expand the Integration Component object, and then select the Integration Component Field object.

The Integration Components and Integration Component Fields lists appear.

9 Inactivate all integration component fields except the following:

- SourceObject Attachment Id, for example, Accnt Attachment Id
- SourceObjectFileExt, for example, AccntFileExt
Siebel EAI and File Attachments ■ Using MIME Messages to Exchange Attachments

■ SourceObjectFileName, for example, AccntFileName
■ Comment

10 Select the SourceObject Attachment Id component field, and then verify that its Data Type property is set to DTYPE_ATTACHMENT.

11 Compile the SRF file and copy it to the object directory under your Siebel Server directory as well as under your Tools directory.

NOTE: Stop the Siebel Server before copying the SRF file. For information on the SRF file, see Using Siebel Tools.

Creating Workflow Examples

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

For more information on creating workflows, see Siebel Business Process Framework: Workflow Guide.

Outbound Workflow

To process the attachment for an outbound request you must create a workflow to query the database, convert the integration object and its attachments into a MIME hierarchy, and then create a MIME document to send to the File Transport business service.

To create an outbound workflow

1 In Siebel Tools, select the Workflow Process object in the Object Explorer.
2 Right-click, then choose New Record.
3 Give the new workflow a name and associate it with a locked project.
4 Right-click, and then choose Edit Workflow Process.

The Workflow Process Designer appears.

5 Create a workflow consisting of Start, End, and four Business Services. Set up each Business Service according to the task it must accomplish.
6 Define your process properties.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Default String</th>
</tr>
</thead>
<tbody>
<tr>
<td>SiebelMessage</td>
<td>Hierarchy</td>
<td>Leave blank.</td>
</tr>
<tr>
<td>Error Message</td>
<td>String</td>
<td>Leave blank.</td>
</tr>
<tr>
<td>Error Code</td>
<td>String</td>
<td>Leave blank.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
<th>Default String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Id</td>
<td>String</td>
<td>Leave blank.</td>
</tr>
<tr>
<td>Process Instance Id</td>
<td>String</td>
<td>Leave blank.</td>
</tr>
<tr>
<td>Siebel Operation Object Id</td>
<td>String</td>
<td>Leave blank.</td>
</tr>
<tr>
<td>MIMEHierarchy</td>
<td>Hierarchy</td>
<td>Leave blank.</td>
</tr>
<tr>
<td>SearchSpec</td>
<td>String</td>
<td>[Account.Name] = 'Sample Account'</td>
</tr>
<tr>
<td>&lt;Value&gt;</td>
<td>String</td>
<td>Default output is binary.</td>
</tr>
</tbody>
</table>

7 The first business service queries the Account information from the database using the EAI Siebel Adapter business service with the Query method. This step requires the following input and output arguments.

<table>
<thead>
<tr>
<th>Input Argument</th>
<th>Type</th>
<th>Value</th>
<th>Property Name</th>
<th>Property Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Integration Object Name</td>
<td>Literal</td>
<td>Sample Account</td>
<td>Not applicable</td>
<td>Not applicable</td>
</tr>
<tr>
<td>SearchSpec</td>
<td>Process Property</td>
<td>Not applicable</td>
<td>SearchSpec</td>
<td>String</td>
</tr>
</tbody>
</table>

**NOTE:** For more information on using the EAI Siebel Adapter, see Chapter 6, “EAI Siebel Adapter Business Service.”
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.

**NOTE:** For more information on the EAI MIME Hierarchy Converter, see "About the EAI MIME Hierarchy Converter" on page 269.

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.

**NOTE:** For more information on the EAI MIME Doc Converter, see "About the EAI MIME Doc Converter" on page 270.

For the final step, set up the last business service of the workflow to write the information into a file using the EAI File Transport business service with the Send method. This step requires the following input arguments.

**NOTE:** For information on File Transport, see Transports and Interfaces: Siebel Enterprise Application Integration.
Inbound Workflow Example

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

NOTE: When passing the process property value for a workflow from an external application (or another business service) as the input property set, the corresponding property name in the input property set must be the same name as the process property and is case sensitive.

To create an inbound workflow

1. In Siebel Tools, select the Workflow Process object in the Object Explorer.
2. Right-click, and then choose New Record.
3. Give the new workflow a name and associate it with a locked project.
4. Right-click, then choose Edit Workflow Process.
   The Workflow Process Designer appears.
5. Create a workflow consisting of Start, End and four Business Services. Set up each Business Service according to the task it must accomplish.
6. Define your process properties.
   Set process properties when you need global properties for the entire workflow:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SiebelMessage</td>
<td>Hierarchy</td>
</tr>
<tr>
<td>Error Message</td>
<td>String</td>
</tr>
<tr>
<td>Error Code</td>
<td>String</td>
</tr>
<tr>
<td>Object Id</td>
<td>String</td>
</tr>
<tr>
<td>Siebel Operation Object Id</td>
<td>String</td>
</tr>
<tr>
<td>MIMEHierarchy</td>
<td>Hierarchy</td>
</tr>
<tr>
<td>MIMEMsg</td>
<td>Binary</td>
</tr>
</tbody>
</table>
7 The first business service in the workflow reads the Account information from a file using the EAI File Transport business service with Receive method. This step requires the following input and output arguments.

<table>
<thead>
<tr>
<th>Input Argument</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name</td>
<td>Literal</td>
<td>c:\temp\account.txt</td>
</tr>
</tbody>
</table>

**NOTE:** For information on File Transport, see Transports and Interfaces: Siebel Enterprise Application Integration.

8 The second business service of the workflow converts the Account information to a MIME hierarchy using the EAI MIME Doc Converter business service with the MIME Doc to MIME Hierarchy method. This step requires the following input and output arguments.

<table>
<thead>
<tr>
<th>Input Argument</th>
<th>Type</th>
<th>Property Name</th>
<th>Property Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIME Message</td>
<td>Process Property</td>
<td>&lt;Value&gt;</td>
<td>Binary</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Output Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIMEHierarchy</td>
<td>Output Argument</td>
<td>MIME Hierarchy</td>
</tr>
</tbody>
</table>

9 The third business service of the workflow converts the MIME hierarchy to a document, and sends it to the EAI Siebel Adapter business service. This step uses the EAI MIME Hierarchy Converter business service with the MIME Hierarchy to Siebel Message method. This step requires the following input and output arguments.

<table>
<thead>
<tr>
<th>Input Argument</th>
<th>Type</th>
<th>Property Name</th>
<th>Property Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIME Hierarchy</td>
<td>Process Property</td>
<td>MIMEHierarchy</td>
<td>Hierarchy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
<th>Output Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>SiebelMessage</td>
<td>Output Argument</td>
<td>Siebel Message</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Input Argument</th>
<th>Type</th>
<th>Property Name</th>
<th>Property Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Siebel Message</td>
<td>Process Property</td>
<td>SiebelMessage</td>
<td>Hierarchy</td>
</tr>
</tbody>
</table>

**About the EAI MIME Hierarchy Converter**

The EAI MIME Hierarchy Converter transforms the Siebel Message into a MIME (Multipurpose Internet Mail Extensions) hierarchy for outbound integration. For inbound integration, it transforms the MIME Hierarchy into a Siebel Message.

**Outbound Integration**

The EAI MIME Hierarchy Converter transforms the input Siebel Message into a MIME Hierarchy. Figure 49 illustrates the Siebel Message of a sample Account with attachments. This figure represents both input and output to the MIME Hierarchy Converter.

![Diagram of Sample Account with Attachments as Input to the MIME Hierarchy Converter](image)

Figure 49. Sample Account with Attachments as Input to the MIME Hierarchy Converter

The output of this process is illustrated in Figure 50.

![Diagram of Output of a MIME Hierarchy Converter](image)

Figure 50. 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. If there is no child of type Attachments in the Siebel Message, then the output is just a MIME Hierarchy with a child of type Document. This document will contain the XML format of the Sample Account integration object instance.

### Inbound Integration

The MIME Hierarchy Converter transforms a MIME Hierarchy input into a Siebel Message. For the inbound process, the first child of the MIME Hierarchy has to be the XML format of the Integration Object instance; otherwise, an error is generated. Figure 51 illustrates the incoming hierarchy.

![Figure 51. Output of a MIME Hierarchy Converter](image)

The output of this process is illustrated in Figure 49 on page 269. The output for this process is the same as the input.

### About the EAI MIME Doc Converter

The MIME Doc Converter converts a MIME Hierarchy into a MIME Message and a MIME Message into a MIME Hierarchy. A MIME Hierarchy contains two different types of property sets.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIME Hierarchy</td>
<td>Mapping to a MIME multi-part</td>
</tr>
<tr>
<td>Document</td>
<td>Mapping to MIME basic-part</td>
</tr>
</tbody>
</table>
**EAI MIME Doc Converter Properties**

Table 47 illustrates some examples of how a MIME Message maps to a MIME Hierarchy.

<table>
<thead>
<tr>
<th>MIME Message</th>
<th>MIME Hierarchy</th>
</tr>
</thead>
</table>
| MIME-Version: 1.0  
Content-Type: application/xml  
Content-Transfer-Encoding: 7bit  
This is a test. | Type: Document  
Value: This is a test |
| MIME-Version: 1.0  
Content-Type: multipart/related; type="application/xml"; boundary=--abc  
----abc  
Content-Type: application/xml  
Content-Transfer-Encoding: 7bit  
This is test2.  
----abc-- | Type: MIMEHierarchy  
Type: Document  
Value: This is a test2 |
The business service needs the following properties on the child property set as shown in Table 48. These properties reflect the most accurate information on the data contained in the child property set.

Table 48. Properties for EAI MIME Doc Converter

<table>
<thead>
<tr>
<th>Property</th>
<th>Possible Values</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContentId</td>
<td>Any value</td>
<td>Document</td>
<td>No Default. The ContentId is the value used to identify the file attachment when the receiver parses the MIME message. When importing attachments, use a unique value for this property and not repeat it for the rest of the file attachments. This is required in the actual document. This property is automatically populated when you are exporting an attachment from a Siebel application.</td>
</tr>
<tr>
<td>Extension</td>
<td>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</td>
<td>Document</td>
<td>No Default. If ContentType and ContentSubType are not defined, then Extension is used to retrieve the appropriate values from this property. If all three values are specified, then ContentType and ContentSubType values override the values retrieved from the Extension. If either the Extension or both ContentType and ContentSubType are not specified, then ContentType will be set to application and ContentSubType will have the value of octet-stream.</td>
</tr>
</tbody>
</table>
Table 48. Properties for EAI MIME Doc Converter

<table>
<thead>
<tr>
<th>Property</th>
<th>Possible Values</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContentType</td>
<td>application, audio, image, text, video</td>
<td>Document</td>
<td>Default is application. The ContentType value has to be specified if you want to set the content type of the document instead of using the extension to get a value from the MIME utility function. If the value is not provided, then the default value is used. The ContentType of multipart is used to represent file attachments in a MIME message. Other values to describe a multipart message are not supported.</td>
</tr>
<tr>
<td>ContentSubType</td>
<td>plain, richtext, html, xml (used with ContentType of Text)</td>
<td>Document</td>
<td>Default is octet-stream. The ContentSubType value has to be specified if you want to set the content subtype of the document instead of using the extension to get a value from the MIME utility function. If the value is not provided, then the default value is used. <strong>NOTE:</strong> Octet-stream is transparent and uses nonencoded 8-bit bytes. The MIME message will contain the binary file data as is, which might cause issues in data transmission over networks that remove bit number 8 (the hi-bit) for special needs.</td>
</tr>
<tr>
<td></td>
<td>octet-stream, pdf, postscript, x-tar, zip, x-javascript, msword, x-conference, x-gzip (used with ContentType of application)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>aiff, basic, wav (used with ContentType of audio)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>gif, jpeg, tiff, x-xbitmap (used with ContentType of image)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>avi, mpeg (used with ContentType of video)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** On the inbound direction, the business service is independent of the transport. It assumes that the input property set contains the MIME message, and writes a property set representation of the MIME message. A property set is used to represent each part of the MIME message. When decoding the MIME message, the business service automatically sets the properties based on the values in the MIME message.
Using Inline XML to Exchange Attachments

To exchange attachments between applications, you use the EAI Siebel Adapter business service:

- To send a message to an external application, call the EAI Siebel Adapter with an integration object that has an integration component from an attachment business component. The EAI Siebel Adapter generates the integration object hierarchy and then converts it to an XML document. The attachment is included in the XML in the `SourceObjectFileBuffer` element.

- To insert an attachment into a Siebel Business Application, the external application uses the same integration object hierarchy, making sure the required fields are present, and puts the base64 string corresponding to the attachment into this message. The XML converter converts the message into an integration object hierarchy, and the EAI Siebel Adapter inserts the attachment.

**NOTE:** Attachments must be in base64 format.

Perform the following tasks to create and test inline XML attachments using an integration object and a workflow:

- Creating an attachment integration object using the EAI Siebel Wizard business service
  For information, see “Creating an Attachment Integration Object” on page 262.
  
  **CAUTION:** To avoid SQL errors, you must inactivate all integration component fields in the integration object except those in Step 9 on page 263.

- “Creating an Attachment” on page 274
- “Creating a Test Workflow” on page 275
- Testing your workflow using the Workflow Simulator
  For information, see *Business Processes and Rules: Siebel Enterprise Application Integration*.

Creating an Attachment

You create an attachment to a record in the Siebel client whose row ID you know.

**To create the attachment**

1. In the Siebel client, navigate to a record that can take an attachment, for example, an account.
2. Choose Help, then About Record from the application-level menu to obtain the row ID of the record.
3. Drill down on the record, then select the Attachments tab.
4. Add an attachment to the record if none exists.
Creating a Test Workflow
You create a workflow in Siebel Tools to do the following:

- Query the Siebel database for the record with the attachment.
- Convert the integration object and its attachment into a Siebel Message.
- Read an external XML file (containing an attachment) and convert it into a Siebel Message.
  **NOTE:** The XML file must use the exact integration object hierarchy as the attachment integration object you created.
- Insert the record into the Siebel database.

**To create a test workflow to exchange attachments**

1. Create the following workflow:

   ![Workflow Diagram]

2. Define the process properties as shown in the following table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Code</td>
<td>String</td>
</tr>
<tr>
<td>Error Message</td>
<td>String</td>
</tr>
<tr>
<td>Object Id</td>
<td>String</td>
</tr>
<tr>
<td>Process Instance Id</td>
<td>String</td>
</tr>
<tr>
<td>Siebel Operation Object Id</td>
<td>String</td>
</tr>
<tr>
<td>SiebelMessage</td>
<td>Hierarchy</td>
</tr>
</tbody>
</table>

---

Integration Platform Technologies: Siebel Enterprise Application
Integration Version 8.1/8.2
The first business service step queries the database using the EAI Siebel Adapter business service with the Query method. This step requires the following input and output arguments:

<table>
<thead>
<tr>
<th>Input Argument</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OutputIntObjectName</td>
<td>Literal</td>
<td>Attachment integration object you created, for example, Account Attachment</td>
</tr>
<tr>
<td>PrimaryRowId</td>
<td>Literal</td>
<td>Row ID of the record to which you added an attachment</td>
</tr>
</tbody>
</table>

The second business service step writes the integration object hierarchy to an XML file using the EAI XML Write to File business service with the WriteEAIMsg method. This step requires the following input arguments:

<table>
<thead>
<tr>
<th>Input Argument</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>Literal</td>
<td>File to write, for example, d:\temp\AttachmentTest_write.xml</td>
</tr>
<tr>
<td>SiebelMessage</td>
<td>Process Property</td>
<td>SiebelMessage</td>
</tr>
</tbody>
</table>

The third business service step reads an XML hierarchy and converts it into a Siebel Message using the EAI XML Read From File business service with the ReadEAIMsg method. This step requires the following input and output arguments:

<table>
<thead>
<tr>
<th>Input Argument</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>Literal</td>
<td>File to read, for example, d:\temp\AttachmentTest_read.xml</td>
</tr>
</tbody>
</table>

**NOTE:** For testing purposes, you can use a modified form of the file written in the second business step, which will automatically have the correct hierarchy.

The fourth business service step reads the Siebel Message and inserts the record into the Siebel database using the EAI Siebel Adapter business service with the Insert method. This step requires the following input argument:

<table>
<thead>
<tr>
<th>Input Argument</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SiebelMessage</td>
<td>Process Property</td>
<td>SiebelMessage</td>
</tr>
</tbody>
</table>
The external business component feature provides a way to access data that resides in a non-Siebel table or view, using a Siebel business component.

This chapter contains the following topics:

- Process of Configuring External Business Components on page 278
- Using Specialized Business Component Methods for EBCs on page 290
- Usage and Restrictions for External Business Components on page 291
- About Using External Business Components with the Siebel Web Clients on page 292
- About Overriding Connection Pooling Parameters for the Data Source on page 293
- About Joins to Tables in External Data Sources on page 293
- About Distributed Joins on page 294
- Troubleshooting External Business Components on page 296

Before You Begin
Before continuing with configuring and implementing external business components (EBCs), review the following books on the Siebel Bookshelf:

- Configuring Siebel Business Applications
- Siebel Developer’s Reference
- Siebel Tools Online Help
- Using Siebel Tools
Process of Configuring External Business Components

Before proceeding, review “Configuring the External Business Component” on page 286. To configure EBCs, you perform the following high-level tasks:

1. "Creating the External Table Definition” on page 279.
   Import the external table definition into Siebel Tools using the External Table Schema Import Wizard.
   This wizard creates a new Table object definition in the Siebel Repository, based upon the contents of a DDL (data definition language) file, or from an Open Database Connectivity (ODBC) data source.

   Map columns in the external table or view to Siebel CRM system fields.
   **NOTE:** One column in the external table or view must be mapped to the Id system field by setting the System Field Mapping property for the column.

   Configure the table definition and specify the data source object.
   The Data Source object is a child object of the Table Object in Siebel Tools and will have to be exposed in the Object Explorer if not already visible. For information on exposing objects in the Object Explorer of Siebel Tools, see *Using Siebel Tools*.
   This object tells the object manager which data source to use to access the object.

   When the table is imported, specify additional table properties for the corresponding external table.

   Configure the EBC and specify the data source object. This data source name will be the same as that specified for the Table object.

   After the data source definition is named in Siebel Tools, specify the run-time parameters by completing the following:
   - Configure the data source definition.
   - Update the server component definition.
Creating the External Table Definition

You use Siebel Tools and the External Table Schema Import Wizard to import your external table definition into the Siebel Repository.

You can create the table definition in one of two ways:

■ “Creating the External Table Definition from a DDL File” on page 279
■ “Creating the External Table Definition from an ODBC Data Source” on page 280

For more information about using Siebel Tools, see Using Siebel Tools.

This task is a step in “Process of Configuring External Business Components” on page 278.

Creating the External Table Definition from a DDL File

You can use the External Table Schema Import Wizard to create the table definition from a data definition language (DDL) file.

It is possible to import an external view definition rather than a table definition. When a view rather than a table definition is imported, it is necessary to amend the Type property of the created Table definition to reflect External View.

NOTE: You can import a database view definition as well as a table definition here. While no difference exists in the resulting Siebel Table object, if it references an external database view, only read access from the Siebel Application is supported.

**To create the external table definition from a DDL file**

1. In Siebel Tools, check out and lock the appropriate project.
2. From the File menu, choose New Object to display the New Object Wizards dialog box.
3. Click the General tab, then double-click External Table Schema Import.
4. In the External Table Schema Import Wizard, specify the following values, then click Next:
   a. Choose the project with which the new Table object definition will be associated.
   b. Specify DDL/Analytics as the schema source type.
   c. Choose the three-digit group code for table name generation. For example, if you choose AXA, then the format of the table names generated will be:
      
      EX_AXA_00000001

5. In the Import External Schema - DDL dialog box, specify the following, then click Next:
   a. Specify the database where the external table resides. The value specified must correspond to the database used by the Siebel schema, for example, Oracle Server Enterprise Edition.
   b. Provide the full path for the location of the SQL/DDL file that contains the external table definition.
6. Confirm the entries, then click Finish to import the DDL file.

A Table object definition is added to the Siebel Repository, corresponding to the external table.
7 Repeat Step 2 through Step 6 for every external table definition you want to import.

Creating the External Table Definition from an ODBC Data Source
You can use the External Table Schema Import Wizard to create the table definition from an Open Database Connectivity (ODBC) data source.

To create the external table definition from an ODBC data source
1 In Siebel Tools, check out and lock the appropriate project.
2 From the File menu, choose New Object to display the New Object Wizards dialog box.
3 Click the General tab, then double-click External Table Schema Import.
4 In the External Table Schema Import Wizard, specify the following values, then click Next:
   a Choose the project with which the new Table object definition will be associated.
   b Specify ODBC as the schema source type.
   c Choose the three-digit group code for table name generation. For example, if you choose AXA, then the format of the table names generated will be:

   EX_AXA_00000001

5 In the next dialog box, click Select Data Source.
   The Select Data Source dialog box appears.
6 Click the Machine Data Source tab, select the appropriate data source name, and then click OK.
7 In the Connect to Database Type dialog box, on the Login tab enter the database user ID and password, then click OK.
8 Enter the table owner for the data source, then click Next.
9 Select the tables to import, then click Next.
10 Confirm the entries, then click Finish.
   A Table object definition is added to the Siebel Repository for each external table selected.

About Data Type Mappings for Importing Table Definitions
When importing table definitions, certain data type mappings are supported for use with the Siebel application. Table 49 contains the data type mappings you can use when importing table definitions.

<table>
<thead>
<tr>
<th>Supported Data Type</th>
<th>Siebel Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MS SQL Server Data Types</td>
<td></td>
</tr>
<tr>
<td>int</td>
<td>Numeric with scale of 0</td>
</tr>
<tr>
<td>bigint</td>
<td>Numeric with scale of 0</td>
</tr>
</tbody>
</table>
**External Business Components**

Process of Configuring External Business Components

### Supported Data Type Mappings by Product

<table>
<thead>
<tr>
<th>Supported Data Type</th>
<th>Siebel Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>smallint</td>
<td>Numeric with scale of 0</td>
</tr>
<tr>
<td>tinyint</td>
<td>Numeric with scale of 0</td>
</tr>
<tr>
<td>float</td>
<td>Numeric</td>
</tr>
<tr>
<td>real</td>
<td>Numeric</td>
</tr>
<tr>
<td>decimal</td>
<td>Numeric</td>
</tr>
<tr>
<td>money</td>
<td>Numeric</td>
</tr>
<tr>
<td>smallmoney</td>
<td>Numeric</td>
</tr>
<tr>
<td>bit</td>
<td>Numeric with a value of 0 or 1</td>
</tr>
<tr>
<td>char</td>
<td>Character</td>
</tr>
<tr>
<td>nchar</td>
<td>Character</td>
</tr>
<tr>
<td>varchar</td>
<td>Varchar</td>
</tr>
<tr>
<td>nvarchar</td>
<td>Varchar</td>
</tr>
<tr>
<td>text</td>
<td>Long</td>
</tr>
<tr>
<td>ntext</td>
<td>Long</td>
</tr>
<tr>
<td>datetime</td>
<td>Date Time</td>
</tr>
<tr>
<td>smalldatetime</td>
<td>Date Time</td>
</tr>
</tbody>
</table>

**DB2 Universal Database Data Types**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT</td>
<td>Numeric with scale of 0</td>
</tr>
<tr>
<td>BIGUINT</td>
<td>Numeric with scale of 0</td>
</tr>
<tr>
<td>SMALLUINT</td>
<td>Numeric with scale of 0</td>
</tr>
<tr>
<td>FLOAT</td>
<td>Numeric</td>
</tr>
<tr>
<td>REAL</td>
<td>Numeric</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>Numeric</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>Numeric</td>
</tr>
<tr>
<td>CHAR</td>
<td>Character</td>
</tr>
<tr>
<td>VARGRAPHIC</td>
<td>Varchar</td>
</tr>
<tr>
<td>LONG VARGRAPHIC</td>
<td>Long</td>
</tr>
<tr>
<td>CLOB</td>
<td>CLOB</td>
</tr>
<tr>
<td>DATE</td>
<td>Datetime</td>
</tr>
<tr>
<td>TIME</td>
<td>Datetime</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>Datetime</td>
</tr>
</tbody>
</table>
### Supported Data Type Mappings by Product

<table>
<thead>
<tr>
<th>Supported Data Type</th>
<th>Siebel Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Oracle Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>Number</td>
<td>Numeric</td>
</tr>
<tr>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>Numeric</td>
</tr>
<tr>
<td>TIMESTAMP WITH LOCAL TIME ZONE</td>
<td>Numeric</td>
</tr>
<tr>
<td>Char</td>
<td>Character</td>
</tr>
<tr>
<td>Nchar</td>
<td>Character</td>
</tr>
<tr>
<td>varchar2</td>
<td>Varchar</td>
</tr>
<tr>
<td>nvarchar2</td>
<td>Varchar</td>
</tr>
<tr>
<td>Long</td>
<td>Long</td>
</tr>
<tr>
<td>CLOB</td>
<td>CLOB</td>
</tr>
<tr>
<td>date</td>
<td>Datetime</td>
</tr>
<tr>
<td><strong>Oracle Business Intelligence (BI) Server Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>Integer</td>
<td>Numeric with scale of 0</td>
</tr>
<tr>
<td>Smallint</td>
<td>Numeric with scale of 0</td>
</tr>
<tr>
<td>Tinyint</td>
<td>Numeric with scale of 0</td>
</tr>
<tr>
<td>Float</td>
<td>Numeric</td>
</tr>
<tr>
<td>Double</td>
<td>Numeric</td>
</tr>
<tr>
<td>Bit</td>
<td>Character (1)</td>
</tr>
<tr>
<td>Boolean</td>
<td>Character (1)</td>
</tr>
<tr>
<td>Char</td>
<td>Character</td>
</tr>
<tr>
<td>Varchar</td>
<td>Varchar</td>
</tr>
<tr>
<td>Longvarchar</td>
<td>Long</td>
</tr>
<tr>
<td>Datetime</td>
<td>Datetime</td>
</tr>
<tr>
<td>Date</td>
<td>Datetime</td>
</tr>
<tr>
<td>Time</td>
<td>Datetime</td>
</tr>
</tbody>
</table>
Table 50 contains the data types that are not supported for importing table definitions.

Table 50. Unsupported Data Type Mappings by Product

<table>
<thead>
<tr>
<th>Database</th>
<th>Unsupported Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>MS SQL Server</td>
<td>timestamp, varbinary,</td>
</tr>
<tr>
<td></td>
<td>binary, image, cursor,</td>
</tr>
<tr>
<td></td>
<td>uniqueidentifier</td>
</tr>
<tr>
<td>DB2 Universal Database</td>
<td>DBCLOB, BLOB</td>
</tr>
<tr>
<td>Oracle</td>
<td>TIMESTAMP, NCLOB, BLOB</td>
</tr>
<tr>
<td></td>
<td>BFILE, ROWID, UROWID,</td>
</tr>
<tr>
<td></td>
<td>RAW, LONG RAW</td>
</tr>
<tr>
<td></td>
<td>INTERVAL YEAR TO MONTH</td>
</tr>
<tr>
<td></td>
<td>INTERVAL DAY TO SECOND</td>
</tr>
<tr>
<td>Oracle BI Server</td>
<td>Timestamp, Varbinary,</td>
</tr>
<tr>
<td></td>
<td>Longvarbinary, Binary,</td>
</tr>
<tr>
<td></td>
<td>Object, Unknown</td>
</tr>
</tbody>
</table>

About the New Imported Table Definition

After the table definition is imported using the External Table Schema Import Wizard, the external table and the external column names are generated.

The external table name is stored in the Table object’s Alias property. This external table name contains the following:
External Business Components ▪ Process of Configuring External Business Components

- An `EX` prefix (for external table)
- A three-digit batch code specified in the External Table Schema Import Wizard
- An automatically generated seven-digit number

An example of the Table name is `EX_ABC_0000001`.

The external column name is stored in the Column child object’s Alias property. An `X` is added as the prefix and a four-digit number is added as the suffix for the external column name, for example, `X_ABC_0000001_0001`.

The Table object’s Type property is set to External or External View (if a view was imported). This column denotes that the table resides outside of the Siebel database.

**Mapping External Columns to Siebel CRM System Fields**

This task is a step in “Process of Configuring External Business Components” on page 278.

When the EBC is defined, you must map the Siebel CRM system fields to the corresponding external table column. System field mapping is accomplished at the column level, rather than using business component user properties. Specify the System Field Mapping column attribute if you want to map a Siebel system field to a column.

**NOTE:** At a minimum, the Id field must be mapped to a unique column defined in the external table and in the Table object definition, which is specified in the business component’s Table property.

By default, the Siebel CRM system fields are not included in the generated SQL for external tables. System Field Mapping is used to specify the mapping between table columns and Siebel CRM system fields. The following is a list of the system fields that can be mapped to external table columns:

- **Conflict Id.** (Optional).
- **Created.** (Optional) Datetime corresponding to when the record was created.
- **Created By.** (Optional) String containing the user name of the person and the system that created the records.
- **Extension Parent Id.** (Optional).
- **Mod Id.** (Optional).
- **Non-system.** (Optional).
- **Updated.** (Optional) Datetime corresponding to when the record was last updated.
- **Updated By.** (Optional) String containing the user name of the person and system that last updated the record.
- **Id.** Mandatory. The single column unique identifier of the record. A column in the external table must be mapped to the Id field.

**NOTE:** The System Field Mapping property must be used in conjunction with external tables only.
Specifying the Data Source Object

This task is a step in “Process of Configuring External Business Components” on page 278.

When the external table has been defined, specify the data source for the corresponding external table. The Data Source child object of the Table object specifies the data source for the corresponding external table:

- The Data Source child object corresponds to a data source defined in the application configuration file (.cfg) or in the Application - Server Configuration screen, Profile Configuration view.
- The Data Source child object instructs the Application Object Manager to use the data source for a specific table. If a Data Source child object is not specified, then the default data source for the application will be used.

**NOTE:** The Data Source child object is specified for external tables only.

For more information about the Data Source child object, see Siebel Tools Online Help.

Specifying Any Optional Table Properties

When the table is imported, you can specify additional table properties for the corresponding external table:

- **External API Write.** Allows you to perform reads directly from the database and have write operations processed by way of a script.
  
  A Boolean property is used to indicate whether or not inserts, updates, or deletes to external tables will be handled by an external API. If this property is set to TRUE, then add scripts to the BusComp_PreWriteRecord and BusComp_PreDeleteRecord events to publish the insert, update, or delete operation to an external API.

- **Key Generation Business Service.** Allows a business service to generate a primary key (Id field) for a business component. If this is not specified, then the Siebel application will generate a row_id value for the column that corresponds to the Id system field.

- **Key Generation Service Method.** Allows a business service method to be called when generating a primary key for a business component.
  
  This method returns a property with the name set to the external table's key column name, and the value set to the generated key:
  
  ```java
  Outputs.SetProperty(<my_external_key_column_name>, <generated_key>);
  ```

  For more information about these table properties, see Siebel Tools Online Help.

This task is a step in “Process of Configuring External Business Components” on page 278.
Configuring the External Business Component

When a Table object definition corresponding to the external table exists in the repository, you can configure a business component to use the new Table object definition.

In general, configuring an EBC is similar to configuring a standard business component with the following exceptions:

- **Data Source business component property.** Specify the Data Source business component property. Set the value for this property to the name of the corresponding Table Data Source.
- **Log Changes property.** Set the Log Changes property to False (unchecked). This will prevent Siebel Remote or Replication transactions from being created. (The default is true.)
- **Intersection table.** When configuring a many-to-many relationship, the intersection table resides in the same database instance as the child table.
- **CSSBusComp class.** It is recommended that all EBCs use the CSSBusComp class.

**NOTE:** Substituting a Siebel-provided table with an external table can result in significant downstream configuration work, and in some cases can restrict or prevent the use of standard functionality provided for the Siebel Business Applications.

This task is a step in “Process of Configuring External Business Components” on page 278.

Specifying Run-Time Parameters

After the data source definition is named in Siebel Tools, you specify the run-time parameters by configuring the data source definition, and updating the server component definition.

If you are testing by using the Siebel Developer Web Client, then add a [DataSource] section to the client .cfg file.

This task is a step in “Process of Configuring External Business Components” on page 278.

Configuring the Data Source Definition

As part of specifying the run-time parameters, configure the data source definition.

**To configure the data source definition**

1. Navigate to the Administration - Server Configuration screen, Enterprises view, then Profile Configuration.
2. Copy an existing InfraDatasources named subsystem type.
3. Change the Profile and Alias properties to the Data Source name configured in Siebel Tools.
4. Update the profile parameters to correspond to the external RDBMS:
   - **DSConnectString = data source connect string**
     - For the Microsoft SQL Server or the IBM DB2 databases, create an ODBC or equivalent connection and input the name of this in the parameter.
For an Oracle RDBMS, this value must specify the TNS name associated with the database and not an ODBC or other entry.

**DSSQLStyle** = *database SQL type*

See Table 51 on page 289 for a listing of the supported SQL types.

**DSDLLName** = *DLL or library name corresponding to the SQL type*

See Table 51 on page 289 for a listing of the supported connector Dynamic Link Library (DLL, Windows) or library (shared object, UNIX) names and SQL styles.

**DSTableOwner** = *data source table owner*

**DSUsername** = *default username used for connections* (Optional)

**DSPassword** = *default password used for connections* (Optional)

**NOTE:** The DSUsername and the DSPassword parameters are optional. However, to avoid receiving a login prompt when accessing the external data source, specify DSUsername and DSPassword. If specified, they will override the default username and password.

The DSUsername and the DSPassword parameters are activated only when using the Database Security Adapter. For more information, see "Configuring a User in LDAP or ADSI Security Adapter To Access EBCs" on page 287.

### Configuring a User in LDAP or ADSI Security Adapter To Access EBCs

You do not typically use Lightweight Directory Access Protocol (LDAP) or Microsoft Active Directory Service Interfaces (ADSI) to retrieve the database username and password to be used for the EBC. Instead, you specify a common DSUsername and DSPassword as described in "Configuring the Data Source Definition" on page 286.

**NOTE:** You do not need to configure LDAP or ADSI authentication unless you need different database users for the EBC (more than one for all users), for example, to have all call center users access the EBC data source with a specific database user, or to be able to set a specific database user for one or more Siebel login IDs.

When LDAP or ADSI authentication is used, the username and password values for the external data source are provided in the ADSI `SharedCredentialsDN` parameter and the `CredentialAttributeType` attribute.

**NOTE:** The `CredentialAttributeType` attribute must be able to hold multiple values, for example, `url`. It cannot be single-value, such as `mail`.

For example, the name of your external data source is `MyExtDataSrc`, and your ADSI is configured with the following parameters:

```
  SharedCredentialsDN= cn=sharedcred,ou=people,dc=siebel,dc=com
  CredentialAttributeType = url
```

In your ADSI server modify the `url` attribute for the following entry:

```
  cn=sharedcred,ou=people,dc=siebel,dc=com
```
External Business Components ■ Process of Configuring External Business Components

Before modifying, one value must already exist (assuming saadmin and db2 are the user name and password for the ServerDataSrc data source, which is the primary data source):

\[
type=ServerDataSrc \ username=saadmin \ password=db2
\]

Add additional values to the url attribute (assuming mmay and mmay are the user name and password for the MyExtDataSrc data source, which is the external data source):

\[
type=MyExtDataSrc \ username=mmay \ password=mmay
\]

After adding the new value for the external data source to the url attribute, you are able to access EBCs.

Configuring the Data Source Definition for the Siebel Developer Web Client

If testing using the Siebel Developer Web Client, then add a [DataSource] section to the client .cfg file for the data source definition named in Siebel Tools. In this example, WindyCity is the data source being added.

**To configure the data source definition in the Siebel Developer Web Client**

1. Add the data source definition named in Siebel Tools. In this example, the data source definition named is WindyCity:

   ```
   [DataSources]
   Local = Local
   Sample = Sample
   ServerDataSrc = Server
   GatewayDataSrc = Gateway
   WindyCity = WindyCity
   ```

2. In the data source section of the application’s .cfg file, add the following parameters (for the supported SQL types and connector DLL names, see Table 51 on page 289):

   ```
   [WindyCity]
   Docked = TRUE
   ConnectString = data source connect string
   SqlStyle = database SQL type
   TableOwner = data source table owner
   DLL = DLL Name corresponding to the SQL type
   DSUsername = user id (Optional)
   DSPassword = password (Optional)
   ```
Supported Connector Names and SQL Styles

When defining the DLL (Windows) or library (UNIX) and SQL files for importing the external schema, the external database being used might not be the same as the Siebel database. Table 51 contains the supported connector DLL and library names and their corresponding SQL styles.

**NOTE:** The extension for the DLL or library name is optional.

Table 51. Supported Connector DLL and Library Names and SQL Styles

<table>
<thead>
<tr>
<th>External Database</th>
<th>DLL Name (Windows)</th>
<th>Library Name (UNIX)</th>
<th>SQL Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM DB2</td>
<td>sscddcli.dll</td>
<td>sscddcli.so</td>
<td>DB2</td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td>sscdm80.dll</td>
<td>Not supported</td>
<td>MSSqIServer</td>
</tr>
<tr>
<td>Oracle. Use for Oracle databases.</td>
<td>sscdo90.dll</td>
<td>sscdo90.so</td>
<td>Oracle</td>
</tr>
<tr>
<td>Oracle. Use for Oracle databases with cost-based optimization.</td>
<td>sscdo90.dll</td>
<td>sscdo90.so</td>
<td>OracleCBO</td>
</tr>
<tr>
<td>SQL Anywhere</td>
<td>sscdw9.dll</td>
<td>sscdw90.so</td>
<td>Watcom</td>
</tr>
</tbody>
</table>

**NOTE:** Use OracleCBO only with an Oracle Database 9i (or later) instance, because the parameter values used are different in earlier versions.

Updating the Server Component to Use the New Data Source

As part of specifying the run-time parameters, update the server component to use the new data source.

*To update the server component to use the new data source*

1. Navigate to the Administration - Server Configuration screen, Enterprises view, then Component Definitions.
2. In the Component Definitions list applet, select your Application Object Manager Component. For example, select the Call Center Object Manager (ENU).
3. Choose Start Reconfiguration from the Menu drop-down list on the Component Definitions list applet.

The Definition State of the component will be set to Reconfiguring. Reselect your application component after selecting the Start Reconfiguring menu item.
4 In the Component Parameters list applet, query for OM - Named Data Source name, and update the Value by adding the alias name of the data source specified in the "To configure the data source definition" topic.

The format of the OM - Named Data Source name parameter is a comma-delimited list of data source aliases. It is recommended that you do not modify the default values, and that you add their new data sources to the preexisting list.

5 After the parameter values are reconfigured, commit the new configuration by selecting Commit Reconfiguration from the Menu drop-down list on the Component Definitions list applet.

The new parameter values are merged at the enterprise level.

To cancel the reconfiguration before it has been committed, select Cancel Reconfiguration from the Menu drop-down list on the Component Definitions list applet.

Using Specialized Business Component Methods for EBCs

The following are the specialized business component methods that are supported for use with EBCs:

- IsNewRecordPending
- GetOldFieldValue
- SetRequeryOnWriteFlag (PreWriteRecord event)
- SetRequeryOnWriteFlag (WriteRecord event)

IsNewRecordPending Business Component Method

This method can be called by using a script in the PreWriteRecord event to determine if the current record is newly created. If the record is a new record, then this method returns the value TRUE.

An example script for the use of this method follows:

```javascript
var isNewRecord = this.InvokeMethod("IsNewRecordPending");
```

GetOldFieldValue Business Component Method

This method can be called by using a script in the PreWriteRecord event to retrieve an old field value if needed. This method takes an input parameter, which must be a valid field name, and returns a string containing the old field value.

An example script for the use of this method follows:

```javascript
var oldLoc = this.InvokeMethod("GetOldFieldValue", "Location");
```
SetRequeryOnWriteFlag (PreWriteRecord event) Business Component Method

In the PreWriteRecord event, this method can be used to put the business component into a mode where the current record refreshes from the data source after the write operation. EBCs typically use this method to refresh database sequencing column values on new record operations. This method takes an input parameter of TRUE or FALSE.

An example script for the use of this method follows:

```javascript
var requery = this.InvokeMethod("SetRequeryOnWriteFlag", "TRUE");
```

SetRequeryOnWriteFlag (WriteRecord event) Business Component Method

In the WriteRecord event, this method informs the object manager that the write operation to the data source is processed by using a script rather than a database connector. At the end of the operation, the business component method, SetRequeryOnWriteFlag, can be called again with the FALSE parameter to reset the requery on write mode, if needed.

An example script for the use of this method follows:

```javascript
var extWrite = this.InvokeMethod("SetRequeryOnWriteFlag", "TRUE");
// insert script here to commit the record via an mechanism channel
var resetWrite = this.InvokeMethod("SetRequeryOnWriteFlag", "FALSE");
```

Usage and Restrictions for External Business Components

The following usage guidelines and restrictions apply to EBCs:

- Creating and populating the external table is the responsibility of the customer. Consult your database administrator for the appropriate method to use.

- EBCs cannot be docked, so they do not apply to mobile users on the Siebel Mobile Web Client. Siebel Remote is not supported.

- EBCs support many-to-many relationships with the limitation that for such relationships the intersection table must be from the same data source as the child business component.

- EBCs cannot be loaded using the Enterprise Integration Manager.

- EBCs rely on the Business Object Layer of the Siebel Architecture. Therefore, EBCs are used only in Siebel Server components using this layer such as the Application Object Manager (for example, the Call Center Object Manager), Workflow Process Manager, and so on. EBCs are not used on components not using this layer, such as Workflow Policies (the Workflow Monitor agent) and Assignment Manager.

- The Id field must be mapped to an underlying column in the external table to support insert, update, delete, and query operations.
Using the Oracle Sequence Object to populate the Id system field is not supported. The value of the Id system field has to be known by the object manager at the record commit time, while the Oracle Sequence Object value is populated by the database server when the change is being processed inside the database.

If the column that was mapped to the Id system field has Primary Key checked, then row ID values are generated by the object manager. Otherwise, a user-entered row ID value is assumed, and the object manager does not generate a row ID value for it.

However, in either configuration, the Primary Key column must not use the Oracle Sequence Object.

EBCs with non-English data require the Siebel S_APP_VER and S_SYS_PREF tables to be present in the external database.

Siebel Business Applications use the UNICD_DATATYPES_FLG column of the S_APP_VER table to indicate whether the database is a Unicode database. A value of 8 means UTF-8, and Y means UTF-16. For Non-Unicode databases, the Enterprise DB Server Code Page system preference is also required to have the correct setting.

For help with creating and populating these tables, contact your Oracle sales representative for Oracle Advanced Customer Services to request assistance from Oracle’s Application Expert Services.

For EBCs that contain multivalue groups, if a primary join is enabled, then both the parent and the child business components must be from the same data source. Multivalue groups are also supported as long as such configuration does not require that a distributed join or a subquery be performed.

Siebel visibility control (for example, ViewMode) is not supported for EBCs.

An external join alias must have the same name as the name used for the external table.

EBCs based on Database views can be used for queries only; updates are not supported.

For EBCs that have a parent-child relationship, their related external tables must not have a foreign key constraint set between them on the external database. If they do have a foreign key constraint, then Copy and Deep Copy functionality will not work.

**NOTE:** Significant configuration effort and changes might be required if you choose to reconfigure a standard Siebel business component on an external table. For example, existing join and link definitions are unlikely to function, because the source fields and underlying columns might not exist in the external table.

### About Using External Business Components with the Siebel Web Clients

If EBCs are used with the Siebel Web or Mobile Web Clients, then new data sources corresponding to the data sources specified for the external tables must be added to the specific Siebel application configuration file. If the user name and password for the external data source are different from the current data source, then a log-in window appears to initiate logging into the external data source.
About Overriding Connection Pooling Parameters for the Data Source

Overriding the connection pooling parameters for the data source is supported. If connection pooling is enabled for the component but not for the data source, then set to zero (0) the following:

- DB Multiplex - Max Number of Shared DB Connections (DSMaxSharedDbConns)
- DB Multiplex - Min Number of Shared DB Connections (DSMinSharedDbConns)
- DB Multiplex - Min Number of Dedicated DB Connections (DSMinTrxDbConns) parameters for the data source

About Joins to Tables in External Data Sources

Joins from business components, based on the default data source to a table in an external data source, are supported in the Siebel application.

Like other joined fields, the fields based on the join to the EBC are read-only.

The limitations for joining business components to tables in an external data source are as follows:

- The source field for the join must be based on a table in the default data source.
- The destination column of the join must be the column mapped to System Field Id.
- Multiple single join specifications are not supported for the join to the external table.
- Reverse navigation (for example, a call to go to the last record) is not supported when fields from multiple data sources are active.

Join Constraints are supported. Joins to more than one external table might be specified. However, increasing the number of external joined data sources can cause degradation in performance.

Searching and Sorting on Fields Joined to External Tables

Fields based on a join to an external table can be searched and sorted. However, limitations do exist. The limitations for searching and sorting on fields joined to an external table follow:

- All fields in the sort specification must either be based on columns in the same external table, or be based on columns in the default data source.
- Named search specifications cannot be set on fields from an external data source.

For information on named search specifications, see the topic on the SetNamedSearch method in Siebel Object Interfaces Reference.

Performance tests are recommended if searching and sorting are permitted on fields based on joins to the external tables. The Siebel application does not have information on the data shape in the external tables. The Siebel application follows a rule-based approach to decide the order in which to query the external tables.
For example, consider the case where there are search and sort specifications on the fields in the Siebel Data Source but none on the fields from the external data source. The Siebel application decides to query the Siebel tables first. Only the rows matching the query specification in the current workset are retrieved from the external data source. As more rows are retrieved from the tables in the Siebel Data Source, the rows from the external data source are also retrieved.

The rules become complex when Search and Sort Specifications are applied to multiple data sources. The rules followed are based on the following requirements:

1. Retrieving the first few rows quickly
2. Shipping the least amount of data between the Siebel and external data sources
3. Eliminating a sort step

Step 2 and Step 3 might produce competing results. In that case, Step 2 takes precedence.

If, as result of the search and sort specifications in effect, then the external table on which the Sort is based is not the driving table, the Siebel application raises an error if more than 1000 rows are retrieved. Refine the query specification in the event of this error.

Directives specified using the Business Component User property **External DataSource Field Priority On Search** to allow hinting of the order in which the tables in the data sources will be queried are supported. These directives can be applied based on a knowledge of the data shape in the Siebel and external tables.

For example, using the following property values:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>External DataSource Field Priority On Search: FieldA</td>
<td>1</td>
</tr>
<tr>
<td>External DataSource Field Priority On Search: FieldB</td>
<td>2</td>
</tr>
</tbody>
</table>

A query on Field A is likely to be selective. If there is a search specification on Field A, then the table that field A is based on is considered the driving table.

A query on Field B is likely to be selective. If there is a search specification on Field B and none on Field A, then the table that field B is based on is considered the driving table.

### About Distributed Joins

Just as join objects can be configured in Siebel Tools and represent a 1:1 relationship between tables resident within the Siebel data model, join objects can be configured to represent a 1:1 relationship with tables external to the Siebel database. A distributed join is a 1:1 relationship between tables that spans two relational data sources. This allows a single, logical record to span multiple data sources. In using distributed joins, the join fields are read-only, and the join specification can consist only of a single field. This federated field support provides the ability for the Object Manager to perform the cross-database join.

Distributed joins are configured the same as standard joins. The query is distributed when the Data Source child object of the table provides a *hint* to the Object Manager (OM) to federate the query.
Configuring Distributed Joins and Federated Fields

To configure distributed joins, you perform the following high-level tasks:

- Implement the external data source (similar to what was done for EBCs).
- The Datasource child object of the Table provides a hint to the object manager to federate the query.
- Create the Join.
- Add the fields to the business component.

To configure distributed joins and federated fields

1. Create the Join point to your external table.
2. Create the Join Specification.
   - This is similar to what you do when creating a standard Siebel join.
3. Add Field to Business Component.
   - Add the fields from the external table to the business component using the join specified.

Usage and Restrictions for Distributed Joins

The following usage guidelines and restrictions apply to distributed joins:

- The source field for the distributed join must be based on a table in the business component’s data source.
- The destination column of the distributed join must be a column mapped to the Id System Field.
- Multiple join specifications are not supported for a distributed join. However, join constraints are supported.
- Inner join is not supported for a distributed join.
- Reverse navigation (for example, a call to go to the last record) is not supported when the fields from multiple data sources are active.
- All fields in the sort specification must be from the same data source.
- All fields in the named search specifications must be from the default data source.
Troubleshooting External Business Components

As you create EBCs, it is recommended that you consider the following steps:

1. Configure EBCs for read and make sure that the data is displayed correctly in the application.
   If the development team feels that some fields require script in order to display correctly then defer the implementation of these fields until testing is complete for a simple read.

2. Add any data transformation script or configuration required in order to provide read access to the more complex fields for display.

3. Configure EBCs for update and make sure that the data is stored correctly in the external database(s) and displayed correctly in the Siebel application.
   Do not add any validation logic to the EBC at this time.

4. Once testing of data update is complete, establish any data transformation configuration or script required to update the fields.
   Make sure that the configuration uses script, which is preferred. However, it is recommended that any data transformation scripts be written on the Pre event.
   Data manipulation configuration and scripts must be attached to Post events.

As part of the troubleshooting process associated with EBCs, increasing the tracing level for a number of component events is suggested.

To increase the tracing level of component events

1. Navigate to the Administration - Server Configuration screen, Servers view, Components, Events, and then select the object manager being used.

2. Change the Log Level for the following Event Types to a higher value (the default is 1).
   Initially a value of 4 is recommended.
   - Task Configuration
   - DBC Log
   - SQL
   - Object Manager DB Connection Operation Log
   - General Object Manager Log
   - Object Manager Session Operation and SetErrorMsg Log
   - Object Manager SRF Operation and SetErrorMsg Log
   - Security Adapter Log

Following this change, restarting the affected components is recommended. With the increase log level, more information is stored in the relevant log files. Reset these values back to 1 when troubleshooting is completed.
Predefined EAI Business Services

Table 52 presents the predefined Siebel EAI business services.

Related Topic
For general information on using business services, see Chapter 4, "Business Services."

<table>
<thead>
<tr>
<th>Business Service</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAI BTS COM Transport</td>
<td>CSSEAIBtsComService</td>
<td>EAI Siebel to BTS COM Transport.</td>
</tr>
<tr>
<td>EAI Data Transformation Engine</td>
<td>CSSDataTransformationEngine</td>
<td>EAI Data Transformation Engine (DTE). For information, see Business Processes and Rules: Siebel Enterprise Application Integration. The display name for this business service is EAI Data Mapping Engine.</td>
</tr>
<tr>
<td>EAI Dispatch Service</td>
<td>CSSEAIDispatchService</td>
<td>Dispatch Service. For information, see Business Processes and Rules: Siebel Enterprise Application Integration.</td>
</tr>
<tr>
<td>EAI DLL Transport</td>
<td>CSSDllTransService</td>
<td>EAI DLL Transport. For information, see Transports and Interfaces: Siebel Enterprise Application Integration.</td>
</tr>
<tr>
<td>EAI HTTP Transport</td>
<td>CSSHTTPTransService</td>
<td>EAI HTTP Outbound Transport. For information, see Transports and Interfaces: Siebel Enterprise Application Integration.</td>
</tr>
<tr>
<td>Business Service</td>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EAI Import Export</td>
<td>CSSEAIImportExportService</td>
<td>EAI Import Export Service. Imports integration objects from, or exports them to, XML files.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>NOTE:</strong> This business service is intended only for Siebel user interface sessions in the Siebel Web Client or Siebel Mobile Web Client. For information on converting integration objects to and from XML files, see the chapter on Siebel XML converters in <em>XML Reference: Siebel Enterprise Application Integration</em>.</td>
</tr>
<tr>
<td>EAI Integration Object to XML Hierarchy Converter</td>
<td>CSSEAIIntObjHierCnvService</td>
<td>EAI Integration Object Hierarchy (also known as SiebelMessage) to XML hierarchy converter service. For information, see <em>XML Reference: Siebel Enterprise Application Integration</em>.</td>
</tr>
<tr>
<td>EAI MIME Doc Converter</td>
<td>CSSEAIMimeService</td>
<td>MIME Document Conversion Service. For information, see Chapter 9, “Siebel EAI and File Attachments.”</td>
</tr>
<tr>
<td>EAI MIME Hierarchy Converter</td>
<td>CSSEAIMimePropSetService</td>
<td>EAI MIME Hierarchy Conversion Service. For information, see Chapter 9, “Siebel EAI and File Attachments.”</td>
</tr>
<tr>
<td>EAI MQSeries Server Transport</td>
<td>CSSMqSrvTransService</td>
<td>EAI MQSeries Server Transport.</td>
</tr>
<tr>
<td>EAI MSMQ Transport</td>
<td>CSSMsmqTransService</td>
<td>EAI MSMQ Transport.</td>
</tr>
<tr>
<td>EAI Null Envelope Service</td>
<td>CSSEAINullEnvelopeService</td>
<td>EAI Null Envelope Service. For information, see <em>XML Reference: Siebel Enterprise Application Integration</em>.</td>
</tr>
<tr>
<td>EAI Query Spec Service</td>
<td>CSSEAIQuerySpecService</td>
<td>Used internally by the EAI Siebel Adapter to convert the SearchSpec method argument as a string to an Integration Object Instance that the EAI Siebel Adapter can use as a Query By Example object.</td>
</tr>
<tr>
<td>EAI Siebel Adapter</td>
<td>CSSEASiebelAdapterService</td>
<td>EAI Siebel Adapter. For information, see Chapter 6, “EAI Siebel Adapter Business Service.”</td>
</tr>
</tbody>
</table>
### Predefined EAI Business Services

<table>
<thead>
<tr>
<th>Business Service</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAI Transaction Service</td>
<td>CSSBeginEndTransactionService</td>
<td>EAI Transaction service for working with Siebel transactions, such as begin and end, to find out whether in transaction.</td>
</tr>
<tr>
<td>EAI UI Data Adapter</td>
<td>CSSEAIUDAdapterService</td>
<td>EAI UI Data Adapter. For information, see Chapter 7, &quot;EAI UI Data Adapter Business Service.&quot;</td>
</tr>
<tr>
<td>EAI XML Converter</td>
<td>CSSEAIXMLCnvService</td>
<td>Converts between XML and EAI messages. For information, see XML Reference: Siebel Enterprise Application Integration.</td>
</tr>
<tr>
<td>EAI XML Read from File</td>
<td>CSSEAIXMLPrtService</td>
<td>Reads an XML file and parses to a property set. For information, see XML Reference: Siebel Enterprise Application Integration.</td>
</tr>
<tr>
<td>EAI XML Write to File</td>
<td>CSSEAIXMLPrtService</td>
<td>Prints a property set to a file as XML. For information, see XML Reference: Siebel Enterprise Application Integration.</td>
</tr>
<tr>
<td>EAI XML XSD Generator</td>
<td>CSSEAIISchXSDService</td>
<td>Used to generate an XSD file from an integration object.</td>
</tr>
<tr>
<td>EAI XSD Wizard</td>
<td>CSSXMLSchemaWizard</td>
<td>Used to create integration objects based on XSD files.</td>
</tr>
<tr>
<td>EAI XSLT Service</td>
<td>CSSXSLTSERVICE</td>
<td>EAI XSL Transformation Service. Supports the Apache Xalan API as the XSLT processor and Xerces as the XML parser.</td>
</tr>
<tr>
<td>Read CSV File</td>
<td>CSSCsvParserService</td>
<td>Converts a CSV file to a property set, and can then convert the property set to XML.</td>
</tr>
<tr>
<td>Siebel Message Envelope</td>
<td>CSSEAIISMEEnvelopeService</td>
<td>EAI Siebel Message Envelope Service. For information, see XML Reference: Siebel Enterprise Application Integration.</td>
</tr>
</tbody>
</table>
### Predefined EAI Business Services

<table>
<thead>
<tr>
<th>Business Service</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Converter</td>
<td>CSSXMLCnvService</td>
<td>Converts between XML documents and arbitrary Property Sets. For information, see XML Reference: Siebel Enterprise Application Integration.</td>
</tr>
<tr>
<td>XML Hierarchy Converter</td>
<td>CSSXMLCnvService</td>
<td>Converts between XML documents and XML Property Set or Arbitrary Property Set. For information, see XML Reference: Siebel Enterprise Application Integration.</td>
</tr>
</tbody>
</table>
Property Set Representation of Integration Objects

This appendix describes the relationship between property sets and integration objects. Property sets are in-memory representations of integration objects. For an overview of property sets, see Using Siebel Tools. It includes the following topics:

- Property Sets and Integration Objects on page 301
- Example Instance of an Account Integration Object on page 304

Property Sets and Integration Objects

Many EAI business services operate on integration object instances. Because business services take property sets as inputs and outputs, it is necessary to represent integration objects as property sets. The mapping of integration objects, components, and fields to property sets is known as the Integration Object Hierarchy.

Using this representation, you can pass a set of integration object instances of a specified type to an EAI business service. You pass the integration object instances as a child property set of the business service method arguments. This property set always has a type of SiebelMessage. You can pass the SiebelMessage property set from one business service to another in a workflow without knowing the internal representation of the integration objects.
Property Set Node Types

When passing integration object instances as the input or output of a business service, you can use property sets to represent different node types, as presented in Table 53.

Table 53. Property Set Node Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Parent</th>
<th>Value of Type Attribute</th>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Method Arguments</td>
<td>Not applicable</td>
<td>Ignored</td>
<td>The properties of this property set contain the service specific parameters, such as PrimaryRowId for the EAI Siebel Adapter.</td>
<td>This is the top-level property set of a business service’s input or output. The properties of this property set contain the service-specific parameters (for example, PrimaryRowId for the EAI Siebel Adapter).</td>
</tr>
<tr>
<td>SiebelMessage</td>
<td>Service Method Arguments</td>
<td>SiebelMessage</td>
<td>The properties of this property set contain header attributes associated with the integration object, for example, IntObjectName.</td>
<td>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 Hierarchy.</td>
</tr>
<tr>
<td>Object List</td>
<td>SiebelMessage</td>
<td>ListOfObjectType</td>
<td>Not used.</td>
<td>This property set identifies the object type that is being represented. The root components of the object instances are children of this property set.</td>
</tr>
</tbody>
</table>
### Table 53. Property Set Node Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Parent</th>
<th>Value of Type Attribute</th>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root Component</td>
<td>Object List</td>
<td>Root Component Name</td>
<td>The property names of the property set represent the field names of the component, and the property values are the field values.</td>
<td>This property set represents the root component of an integration object instance.</td>
</tr>
<tr>
<td>Child Component Type</td>
<td>Root Component or Component</td>
<td>ListOfComponent Name</td>
<td>Not used.</td>
<td>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.</td>
</tr>
<tr>
<td>Child Components</td>
<td>Child Component Type</td>
<td>Component Name</td>
<td>The property names of the property set represent the field names of the component, and the property values are the field values.</td>
<td>This property set represents a component instance. It is a grandchild of the parent component’s property set.</td>
</tr>
</tbody>
</table>
Example Instance of an Account Integration Object

This example shows an Account integration object in which the object has two component types: Account and Business Address (which is a child of Account). The hierarchy of component types, from the perspective of Oracle’s Siebel Tools, looks like that shown in Figure 52.

Figure 52. Sample Account Integration Object

Figure 53 on page 305 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.
Figure 53. Partial Instance of Sample Account Integration Object
Property Set Representation of Integration Objects  ■ Example Instance of an Account Integration Object
This appendix lists the various inbound and outbound Document Type Definitions (DTDs) for the XML Gateway business service. It covers the following topics:

- Outbound DTDs for the XML Gateway Business Service on page 307
- Inbound DTDs for the XML Gateway Business Service on page 309

**Outbound DTDs for the XML Gateway Business Service**

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

**Delete**

The following DTD is for the Delete request:

```xml
<!ELEMENT siebel-xml-ext-delete-req (buscomp, remote-source, row)>
<!ELEMENT buscomp (#PCDATA)>
<!ATTLIST buscomp id NMTOKEN #REQUIRED>
<!ELEMENT remote-source (#PCDATA)*> 
<!ELEMENT row (value+)>
<!ELEMENT value (#PCDATA)*> 
<!ATTLIST value field CDATA #REQUIRED>
```

**Init**

The following DTD is for the Init request:

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

**Insert**

The following DTD is for the Insert request:

```xml
<!ELEMENT siebel-xml-ext-fields-req (buscomp, remote-source?)>
<!ELEMENT buscomp (#PCDATA)>
<!ATTLIST buscomp id NMTOKEN #REQUIRED >
<!ELEMENT remote-source (#PCDATA)*> 
```
**PreInsert**
The following DTD is for the PreInsert request:

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

**Query**
The following DTD is for the Query request:

```xml
<!ELEMENT siebel-xmlext-query-req (buscomp, remote-source?, max-rows?, search-string?, match?, search-spec?, sort-spec?)>
<!ELEMENT buscomp (#PCDATA)>
<!ATTLIST buscomp id NMTOKEN #REQUIRED>
<!ELEMENT remote-source (#PCDATA)*>
<!ELEMENT max-rows (#PCDATA)>
<!ELEMENT search-string (#PCDATA)>
<!ELEMENT match (#PCDATA)>
<!ATTLIST match field CDATA #REQUIRED>
<!ELEMENT search-spec (node)>
<!ELEMENT node (#PCDATA | node)*>
<!ATTLIST node node-type (Constant | Identifier | Unary Operator | Binary Operator) #REQUIRED>
<!ATTLIST node value-type (TEXT | NUMBER | DATETIME | UTCDATETIME | DATE | TIME) #IMPLIED>
```
<!ELEMENT sort-spec (sort+)>
<!ELEMENT sort (#PCDATA)>
<!ATTLIST sort field CDATA #REQUIRED>

**Update**
The following DTD is for the Update request:

```
<!ELEMENT siebel-xmlext-update-req (buscomp, remote-source?, row)>
<!ELEMENT buscomp (#PCDATA)>
<!ATTLIST buscomp id NMTOKEN #REQUIRED>
<!ELEMENT remote-source (#PCDATA)*>
<!ELEMENT row (value+)>
<!ELEMENT value (#PCDATA)*>
<!ATTLIST value changed (true | false) #REQUIRED>
<!ATTLIST value field CDATA #REQUIRED>
```

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

**Delete Response**
The following DTD is for the Delete response:

```
<!ELEMENT siebel-xmlext-delete-ret EMPTY >
```

**Init Response**
The following DTD is for the Init response:

```
<!ELEMENT siebel-xmlext-fields-ret (support+)>
<!ELEMENT support EMPTY >
<!ATTLIST support field CDATA #REQUIRED>
```

**Insert Response**
The following DTD is for the Insert response:
PreInsert Response
The following DTD is for the PreInsert response:

```xml
<!ELEMENT siebel-xmlext-preinsert-ret (row)>
<!ELEMENT row (value)+>
<!ELEMENT value (#PCDATA)+>
<!ATTLIST value field CDATA #REQUIRED >
```

Query Response
The following DTD is for the Query response:

```xml
<!ELEMENT siebel-xmlext-query-ret (row*)>
<!ELEMENT row (value+>
<!ELEMENT value (#PCDATA)*>
<!ATTLIST value field CDATA #REQUIRED >
```

Update Response
The following DTD is for the Update response:

```xml
<!ELEMENT siebel-xmlext-update-ret (row)>
<!ELEMENT row (value+>
<!ELEMENT value (#PCDATA)>
<!ATTLIST value field CDATA #REQUIRED >
```
Symbols
%methodName% request, sample outbound DTDs 307
%methodName% response, sample inbound DTDs 309
* (asterisk), using as querying wildcard 147

A
activating fields, about 44
AllowedIntObjects business service user property 40
application
   external application, about setting up 234
arguments
   Init method, XML Gateway business service 225
   IsPrimaryMVG 175
AssocFieldName user property
   associations with 22
Association user property
   associations with 22
association, defined 22

B
base object types, for integration components (table) 15
base table, using Mod Id 192
body data, contents of 14
buscomp Id tag 227
Business Component Id argument, XML Gateway method 225
Business Component Name argument, XML Gateway method 226
business components
   association, role of 22
   integration restrictions 70
   linking 25
   multivalue field example 23
   multivalue group example 26
   relation to business services 73
   specialized 221
business objects
   business service methods, as arguments to 84
   EAI Siebel Adapter business service, role of 145
   external data, creating from 146
   integration object maintenance, about 60
   relation to business services 73
   structure of 18
   user key requirement 32

business service methods
arguments, defining 77
business objects as arguments 84
defining 77
described 74
Business Service Methods screen, using 79
business service methods, custom
   about 234
   common input parameters (table) 235
   connecting methods, list of 234
   Delete method, example 236
   Error Return property set, example 238
   Init method, example 238
   Insert method, example 240
   output parameters (table) 235
   PreInsert method, example 242
   Query method, example 243
   Update method, example 247

Business Service Simulator, running 81
business services
   accessing using Siebel eScript or Siebel VB 82
   argument types 78
customized business services, type of 74
defined 73
deploying as Web services 80
deploying to run-time database from Data Access Service wizard 217
deploying to run-time database from WSDL Import Wizard 101
EAI MIME Hierarchy Converter, creating inbound workflow (example) 268
EAI MIME Hierarchy Converter, creating outbound workflow (example) 266
EAI Siebel Adapter, about 145
EAI Siebel Wizard, about 42
EAI UI Data Adapter, about 197
general uses 73
importing and exporting in Siebel Tools 81
importing into the Siebel application 81
predefined business services, table of 299
property set code example 83
property sets, about and role of 75
scripts, writing 78
Siebel Business Application, creating in 79
Siebel Tools, creating process overview 76
Siebel Tools, defining in 76
Specialized Business Services, about 74
testing 81
user properties, defining 78
XML Gateway 224

BusObjCacheSize argument, about 171, 172, 215

C

calculated fields 29
CamelCase, choosing naming convention 47
child integration components
about 18
structure example 19
child property sets, about 75
classes
associated with predefined EAI business
services (table) 297
CSSBCVExtern 222
CSSBCVXMLExtLen 225
CSSEAIDTEScriptService 74
components, defined 13
concurrency control
about support for 191
Account_Organization integration component
example 195
configuring 193
configuring example 194
Modification IDs, using 192
Modification Key, about 191

ContentId property, value and
description 272
ContentSubType property 273
ContentType property 273
CSSEAISiebelAdapterService class 40
CSSBCVExtern class 222
CSSBCVXMLExtLen class 225
CSSDataTransformationEngine class 297
CSSDIIITransService class 297
CSSEAIBtsComService class 297
CSSEAIDispatchService class 297
CSSEAIDTEScriptService class 74
CSSEAIBtsExportExportService class 298
CSSEAIBtsIntObjHierCnvService class 298
CSSEAIMimePropSetService class 298
CSSEAIMimeService class 298
CSSEAINullEnvelopeService class 298
CSSEAISqlQueryService class 298
CSSEAISiebelAdapterService class 298
CSSEAISMEnvelopeService class 299
CSSEAISUDDAdapterService class 299
CSSEAIXMLCnvService class 299
CSSEAIXMPLPrService class 299
CSSHTTPTransService class 297
CSSMqSrvTransService class 298
CSSMsmqTransService class 298
CSSXMLCnvService class
XML Converter business service 300
XML Hierarchy Converted business
service 300
CSSXMLCnvService class 299
custom business service
Delete method, example 236
sample code, eScript 250
sample code, VB 257

D

Data Access Service wizard, using to expose
Siebel data to external applications 216
data and arguments, contrasted 83
data definition language (DDL) file, creating
an external table definition 279

Data Type Definitions
See DTDs
databases
access, controlling 40
multivalued attributes 22
Delete business service method
custom business service example 236
DTD example 307
overview 160
XML code example 161
Delete Response method, DTD example 309
DeleteByUserKey argument, about 171, 172
DeleteLeaves method, EAI UI Data Adapter
business service 212
Display Name field 75
docking, restrictions on 221
DOC-literal support 90

DTDs
Integration Object Builder wizard, about 17
sample inbound DTDs 309
sample outbound DTDs 307

E

EAI BTS COM Transport business
service 297
EAI Data Mapping Engine business
service 297
EAI Design project, editing integration
objects, caution 18  
EAI Dispatch Service business service 297  
EAI DLL Transport business service 297  
EAI file streaming  
about 140  
criteria 142  
enabling 140, 142  
inbound requests 140  
outbound responses 141  
EAI HTTP Transport  
business service, description 297  
XML Gateway business service, configuring for use by 224  
EAI Import Export business service 298  
EAI Integration Object to XML Hierarchy Converter business service 298  
EAI MIME Doc Converter business service 298  
EAI MIME Hierarchy Converter business service 298  
EAI MQSeries Server Transport business service 298  
EAI MQSeries Transport, configuring for use by XML Gateway business service 224  
EAI MSMQ Transport business service 298  
EAI MSMQ Transport, configuring for use by XML Gateway business service 224  
EAI Query Spec Service business service 298  
EAI Siebel Adapter business service  
about 145  
concurrency control, about support for 191  
database access, controlling 40  
Delete method 160  
Execute method, overview 160  
Insert method, overview 157  
IsPrimaryMVG argument 175  
language-independent code, using 185  
method arguments (table) 171  
methods, list of 146  
Modification IDs, using 192  
Modification Key, about 191  
multivalue groups 175  
predefined EAI business services, table of 298  
QueryPage method, overview 148  
runtime events, about using 186  
Synchronize method, overview 150  
Update method, overview 159  
Upsert method, overview 159  
XML code example 161  
EAI Siebel Wizard business service  
about 42  
integration objects, creating 45  
EAI Transaction Service business service 299  
EAI UI Data Adapter business service  
about 197  
DeleteLeaves method 212  
Execute method 213  
exposing Siebel data to external applications 216  
InitLeaves method 206  
InsertLeaves method 208  
method arguments (table) 215  
method arguments, table of 199  
methods, list of 199  
predefined EAI business services, table of 299  
QueryPage method 199  
UpdateLeaves method 204  
EAI XML Converter business service 299  
EAI XML Read from File business service 299  
EAI XML Write to File business service 299  
EAI XSLT Service business service 299  
EnableExtServiceOnly parameter, eapps.cfg file 142  
Error Return property set example 238  
ErrorOnNonExistingDelete argument, about 171, 172  
error-text tag 231  
eScript  
See scripts, Siebel eScript  
Execute business service method  
EAI UI Data Adapter business service 213  
overview 160  
Execute method  
operations (table) 161  
specifying and supported parent and child components (table) 162  
ExecutionMode argument  
about 171  
ExecutionMode argument, about 173, 215  
Extensible Markup Language (XML) standard 88  
Extensible Stylesheet Language Transformation (XSLT) standard 88  
Extension property, value and description 272  
extension table, using Mod Id 192  
external application  
sample inbound DTDs 309  
sample outbound DTDs 307  
setting up, about 234  
external business components (EBCs)  
configuration process 278
configuring 286
creating external table definition 279
distributed joins 293, 294
distributed joins, configuring 295
distributed joins, usage and restrictions 295
joins to tables in external data sources 293
overriding connection parameters for the data source 293
troubleshooting 296
usage and restrictions 291
using specialized business component methods 290
using with Siebel Web Client 292
external data source, specifying 223
External Name user property 22
External Table Schema Import Wizard, creating external table definitions 279

F
faults, SOAP 131
field dependencies, defining 30
fields
activating and inactivating 44
calculated 29
definition 13
integration component fields, viewing 47
multivalue groups, working with 25
picklist, validating and example 28
property set fields 75
user keys, about 32
file attachments
about and using 261
message types 261
file streaming
about 140
criteria 142
enabling 140, 142
inbound requests 140
outbound responses 141
force active fields, performance considerations 69
function code sample 84

H
header data, contents of 14
hierarchy parent key, about and example 39
hierarchy root key, about and example 39
Hypertext Transfer Protocol (HTTP) standard 88

I
inactivating fields, about 44
incoming XML format, tags and descriptions (table) 231
Init method, DTD example 307
Init property set example 238
Init Response method, DTD example 309
InitLeaves business service method
EAI UI Data Adapter business service 206
inline XML attachments
about 262
using 274
inner joins, and integration components 29
input parameters, common (table) 235
Input/Output type 78
Insert business service method
EAI Siebel Adapter business service 157
Insert business service method, DTD example 307
Insert property set example 240
InsertLeaves business service method
EAI UI Data Adapter business service 208
instance, defined 14
integration component fields
custom 21
defined 14
field names, assigning 27
multivalue groups, working with 25
user properties 28, 30
integration component keys
See user keys
integration component user properties
as operation controls 31
REPOBJ user property 30
integration components
activating 43
component fields, viewing 47
defined 14
deleting during synchronization 58
guidelines and scenarios 70
inner joins 29
multivalue groups, working with 25
operation controls 31
selecting 46
user properties 31
integration messages
body data 14
defined 13
header data 14
Integration Object Builder wizard
about 17
Code Generator wizard 17
EAI Siebel Wizard 43
Generate XML Schema wizard 17
integration components, selecting 46
integration objects, creating 45
user keys, about building  32
user keys, validating  33
integration object instance
  actual data, about and diagram  16
defined  14
integration objects
  about  14
  base object types (table)  15
calculated fields  29
  child integration components  18
  Container Naming Convention menu  46
creating  45
  creating programmatically  49
creating, based on another root business component  48
creating, with many-to-many business component  48
defined  13
deploying to run-time database from Data Access Service wizard  217
deploying to run-time database from EAI Siebel Wizard  47
deploying to run-time database from Siebel Tools  53
deploying to run-time database from WSDL Import Wizard  101
EAI Design project, editing caution  18
external data, creating from  146
fine-tuning practices, list of  52
in-memory updating  56
instance example  304
integration components, deleting during synchronization  58
Lower CamelCase for XML tags check box  47
maintaining, about  60
metadata, about synchronizing  54
metadata, relation to  16
MIME message objects, creating  262
  passing custom attributes  21
performance considerations  69
picklist, validating and example  28
  primaries, about setting  27
property set representation  301
  removing from run-time database  54
schema, generating  68
  simple hierarchy example  304
structure example  19
synchronizing with business objects  61
System fields, about treatment of  70
terminology  13
testing newly created integration object  53
undeploying from run-time database  54
updating  61
UseFormattedValues user property  67
validating  52
  wizards process diagram  17
integration projects
  integration objects, use described  18
  planning  14
IntObjectName argument
  described  173
  locating arguments for  171
IsPrimaryMVG argument  175

J
Java class files, generating  17
joined table, using Mod Id  192

L
language-independent code
  list of values, types of  186
  outbound and inbound direction, about using  185
LargeFileChunkingEnabled parameter, eapps.cfg file  140, 142
LastPage argument, about links
  associations, and  171, 173
  between business components  25
local business services
  about and using  107
  script example  108
  using in outbound Web service  110
LOVLanguageMode argument, about LOVs, language-independent code translation  186

M
many-to-many relationships, virtual business components  221
MessageId argument
  described  173
  locating arguments for  171
metadata
  defined  14
  integration objects, updating  61
  processing example  83
  relation to integration objects  16
  synchronizing, integration objects, about  54
method arguments
  EAI Siebel Adapter business service (table)  171
  EAI UI Data Adapter business service (table)  215
methods
  business objects as arguments  84
  business service method arguments,
Index ■ N

defining 77
business service method arguments, types of 78
business services methods, about 74
business services methods, defining 77
EAI Siebel Adapter business service, supported methods 146
EAI UI Data Adapter business service, supported methods 199
incoming XML tags by method 231
outgoing XML tags by method 227
XML Gateway business service method arguments (table) 225
XML Gateway business service methods, listed 225

MIME
about 261
EAI MIME Doc Converter properties (table) 271
inbound workflow, creating (example) 267
integration objects, creating 262
messages and hierarchies 270
MIME hierarchy, converting to 268
outbound workflow, creating (example) 264
workflow process properties, creating an inbound workflow 267
workflow process properties, creating an outbound workflow 264

MIME Doc Converter
about 270
converting hierarchy to document 266
converting to a hierarchy 268
EAI MIME Doc Converter properties (table) 271
properties 273

MIME hierarchy
converting hierarchy to document 266
converting to a hierarchy 268
EAI MIME Doc Converter properties (table) 271
inbound transformation 270
integration object, converting to MIME hierarchy 266
MIME Doc Converter 270
outbound transformation 269
property sets 270

MIME Hierarchy Converter
business service, creating inbound workflow (example) 268
business service, creating outbound workflow (example) 266
inbound transformation 270
outbound transformation 269

mobile users and virtual business components 221

Modification Key
about 191
Account_Organization integration component example 195
Mod Id field, using for tables 192
MVG and MVGAssociation integration components, configuring 193
MVG and MVGAssociation integration components, configuring example 194

Multi Value Link field 24

Multipurpose Internet Mail Extensions
See MIME

multivalue groups
EAI Siebel Adapter business service, overview 175
element 23
field names, assigning 27
integration components, creating 25
mapping to separate integration objects 175
multiple fields 25
primary record, setting 176
types of 22
virtual business components, restriction 221

multivalue links, setting primaries 27
multivalued attributes 22

MVG
See multivalue groups

MVG integration components
Account_Organization integration component example 195
configuring for concurrency control 193
element 194

MVGAssociation integration components
Account_Organization integration component example 195
configuring for concurrency control 193
element 194

MVGAssociation user property
about 22
MVG, creating a Siebel integration component
to represent 26

N
NamedSearchSpec argument, about 215
name-value pairs
concatenating 224
role in property sets 75

NewQuery argument 173
NewQuery argument, about 215

No envelope business service 298
None session type, SOAP header
examples 127
NumOutputObjects argument
described 173
locating arguments for 171
NumOutputObjects argument, about 216

O
Open Database Connectivity (ODBC) data
source, creating an external table
definition 280
operations, relationship to Web service port
types 93
outgoing XML format, tags and descriptions
(table) 227
Output Integration Object Name argument,
about 173
output parameters, (table) 235
Output type 78
OutputIntObjectName argument,
about 171, 216

P
PageSize
EAI Siebel Adapter business service method
argument 173
locating arguments for 171
parameters
common input parameters (table) 235
output parameters (table) 235
Parameters argument, XML Gateway
method 226
parent business component
multivalue group example 25
multivalue group field names, assigning 27
parent integration component
about 18
identifying 47
structure example 19
performance
force-active fields, considerations 69
integration object considerations 69
picklist considerations 69
picklists
performance considerations 69
validating, about and example 28
port types, relationship to Web service
operations 93
PreInsert method, DTD example 308
PreInsert property set example 242
PreInsert Response method, DTD
element 309, 310
primaries, about setting 27
primary business component 18
primary integration component
See parent integration component
PrimaryRowId argument
described 174
locating arguments for 172
property sets
about 301
about and role of 75
child 75
code sample 83
Delete method example 236
Display Name field 75
EAI MIME Doc Converter properties
(table) 271
Error Return example 238
fields 75
hierarchy example 304
Init example 238
Insert example 240
integration objects, and 301
MIME hierarchy 270
nodes types (table) 302
PreInsert example 242
Query example 243
Update example 247

Q
Query method
business component records, about querying
all 147
DTD example 308
wildcard querying, about using asterisk
(*) 147
query operation
integration component keys, role of 32
role in integration projects 18
Query property set example 243
Query Response method, DTD example 310
QueryByKeyUserKey argument, DTD example
172, 174
QueryPage method
EAI UI Data Adapter business service 199
overview 148

R
Remote Source argument, XML Gateway
method 225
Remote Source user property
virtual business component 223
XML Gateway business service 224
repository objects, REPOBJ user
property 30
REPOSITORY_BC_VIEWMODE_TYPE 40
root component
See parent integration component
row tag 233
RPC-literal support 89
run-time database
deploying business services to 101, 217
deploying integration objects to 47, 53, 101, 217
run-time events, about using 186

S
schema
Generate XML wizard 17
generating 68
scripts
business service, attaching to 78
business service, using to access 82
SearchSpec argument
described 174
locating arguments for 172
ServerDetermine session type, about 122
Service Name user property
virtual business component 222
XML Gateway business service 224
Service Parameters user properties, table of 224
Service Parameters user property
virtual business component 222
XML Gateway business service 224
session timeout parameters
about and table 126
SessionTimeout 126
SessionTokenMaxAge 126
SessionTokenTimeout 126
Siebel Business Application, defining business services 79
Siebel business component, defined 14
Siebel business objects
defined 14
structure of 18
Siebel Developer Web Client, invoking Web services 95
Siebel EAI
See individual EAI entries
Siebel eScript
using to access a business service 82
using to invoke Web services 113
Siebel integration components
See integration components
Siebel integration objects
See integration objects
Siebel Message envelope business service 299
Siebel Message object
See integration object instance
Siebel Mobile Web Client, invoking Web services 95
Siebel Tools
business services, creating process overview 76
business services, defining 76
integration objects, creating 45
user key, identifying 32
virtual business component, creating 222
Siebel VB, using to access a business service 82
Siebel Web Server Extension (SWSE) 120, 130
enabling Session Management 125
SWSE Plug-in 140, 141
SiebelMessage argument
EAI Siebel Adapter business service method argument 174
locating arguments for 172
SiebelMessage argument, about 216
siebel-xmlext-fields-req tag 227
siebel-xmlext-fields-ret tag 232
siebel-xmlext-Insert-req tag 228
siebel-xmlext-insert-ret tag 232
siebel-xmlext-preinsert-req tag 228
siebel-xmlext-preinsert-ret tag 232
siebel-xmlext-query-req tag 229
siebel-xmlext-query-ret tag 233
siebel-xmlext-status tag 231
siebel-xmlext-Update-req tag 230
siebel-xmlext-Update-ret tag 234
Simple Object Access Protocol (SOAP)
about 88
custom filters 136
header examples 127
Siebel Authentication and Session Management headers (table) 119
standard 88
simulation, business service 81
SOAP faults
handling in version 8.1 and higher 135
handling in versions 8.0 and earlier 134
schema support 131
SortSpec argument
EAI Siebel Adapter business service method argument 174
locating arguments for 172
Specialized Business Services, about 74
StartRowNum argument
EAI Siebel Adapter business service method argument 174
locating arguments for 172
Stateful session type, SOAP header
example 128
Stateless session type, SOAP header example 127
status keys, about and example 36
status-code tag 231
StatusObject argument
described 174
locating arguments for 172
SWSE
See Siebel Web Server Extension (SWSE)
synchronization process
about 55
in-memory updating 56
integration object components, deleting 58
integration objects, updating 61
role in integration projects 18
update rules, about 56
Synchronize business service method,
overview 150
System fields, about treatment of 70
T
tables, using Mod Id 192
testing business services 81
transports, used with XML Gateway 224
U
UI Data Sync (UDS) Web Services, about and
using Data Access Service wizard 216
Update business service method,
overview 159
Update Leaves business service method, EAI
UI Data Adapter business service 204
Update method, DTD example 309
Update property set example 247
Update Response method, DTD example 310
Upsert business service method
overview 159
XML code example 161
UseFormattedValues integration object user property 67
user keys
building and validating, example 33
defined 32
definitions, confirming after integration object creation 42
field in Siebel Tools 32
hierarchy parent key, about and example 39
hierarchy root key, about and example 39
inactivating, caution 36
Integration Component key 32
locating in Tables screen 33
Object Builder wizard, about building with 32
status keys, about and example 36
validity, checking 34
user properties
AssocFieldName 22
Association 22
business service user properties, defining 78
External Name 22
MVGAssociation 22
UseFormattedValues 67
virtual business components (table) 222
virtual business components, defining for 223
V
value tag 233
VBC Compatibility Mode user property 223, 224
VBCs
See virtual business components
ViewMode argument
EAI Siebel Adapter business service method
argument 174
locating arguments for 172
ViewMode argument, about 216
ViewMode integration object user property 40
virtual business components
about 219
custom code example, eScript 250
custom code example, VB 257
docking restrictions 221
external application setup, about 234
incoming XML format, tags and descriptions (table) 231
mobile users, restriction 221
MQSeries, implementing with 224
multivalue groups 221
new virtual business component, creating 222
outgoing XML format, tags and descriptions (table) 227
specialized business components, restriction 221
user properties (table) 222
user properties, defining 223
XML Gateway business service, configuring 224
virtual business components, methods
Delete method example 236
Error Return property set, example 238
Init method, example 238
Insert method, example 240
PreInsert property set, example 242
Query property set, example 243
Update property set, example 247

**virtual business services**
See business service methods

**W**

**Web services**
about 87
cache refresh 142
consuming external Web services 100
custom SOAP filters 130, 136
defining Web Service Inbound Dispatcher 94
deploying business services as 80
DOC-literal support 90
EAI file streaming 140
enabling tracing 143
Extensible Markup Language (XML) standard 88
Extensible Stylesheet Language Transformation (XSLT) standard 88
generating a WSDL file 93
Hypertext Transfer Protocol (HTTP) standard 88
inbound, about port types and operations 93
invoking on the Siebel Developer Web Client 95
invoking on the Siebel Mobile Web Client 95
invoking using an external system 91
invoking, examples of 111, 112
Local Business Service 107
one-way operations 90
outbound, creating based on WSDL file 100
outbound, creating manually 102
outbound, importing run-time data 102
outbound, integration objects as input arguments 104
outbound, using local business service 110
publishing inbound 91
RPC-literal support 89
security support 117
Session Management SOAP headers (table) 119
Siebel Authentication 119
Simple Object Access Protocol (SOAP) standard 88
single sign-on authentication 130
SOAP fault message example 95
SOAP fault schema support 131
support for transport headers 106
support for transport parameters 106
using Data Access Service wizard to expose Siebel data to external applications 216
Web Services Description Language (WSDL) standard 88
Web Services Interoperability (WS-I) standard 88
Web Services Security (WS-Security) standard 88
XML Schema standard 88
XML schema support for the xsd:any tag 111

**Web Services Description Language (WSDL)**
about 87
consuming an external Web service using the WSDL Import Wizard 100
generating a WSDL file 93
importing a WSDL file 100
invoking an external Web service 112
standard 88

**Web Services Interoperability (WS-I)**
standard 88

**Web Services Security (WS-Security)**
standard 88
about and reference 88
support for UserName Token mechanism 118

**white space, handling in XML documents** 226

**workflows**
inbound MIME request 267
outbound MIME request 264
policies, about using 186

**WSDL Import Wizard**
deploying business services 101
deploying integration objects 101
invoking an external Web service 112
using to consume an external Web service 100

**X**

**XML**
attribute-named operation, specifying 162
business services, importing and exporting 81
Generate XML Schema wizard 17
handling white space 226
inline XML attachments 262, 274
metadata example 83
upsert and delete code example 161

**XML Converter business service** 300

**XML format**
incoming tags and descriptions (table) 231
outgoing tags and descriptions (table) 227
XML Gateway business service
about 224
configuring 224
incoming XML tags and descriptions 231
init method arguments 225
methods (table) 225
methods arguments (table) 225
name-value pairs, concatenating 224
outgoing XML tags and descriptions 227
sample inbound DTDs 309

sample outbound DTDs 307
Virtual Business Component, implementing with MQSeries 224

XML Hierarchy Converter business service 300

XML Schema standard 88

XSD container elements, choosing naming convention 46
xsd:any tag, XML schema support 111