Integration Platform Technologies: Siebel Enterprise Application Integration

Version 7.8, Rev. B

December 2008

ORACLE

Contents

Chapter 4: Business Services

Chapter 5: Web Services

Chapter 6: EAI Siebel Adapter

Chapter 9: Siebel EAI and File Attachments

Chapter 10: External Business Components

1 What's New in This Release

What's New in Integration Platform Technologies: Siebel Enterprise Application Integration, Version 7.8, Rev. B

Table 1 lists changes described in this version of the documentation to support release 7.8 of Oracle's Siebel software.

Table 1. New Product Features in Integration Platform Technologies: Siebel Enterprise Application Integration, Version 7.8, Rev. B

Topic	Description
"Search Specification" on page 155	Added the following to the list of supported operators for the searchspec attribute of the QueryPage method of the EAI UI Data Adapter business service: ■ Parentheses, including nested parentheses ■ EXISTS, NOT EXISTS ■ IS NULL, IS NOT NULL
"Creating an Attachment Integration Object" on page 210	When creating an attachment integration object, keep only certain integration component fields active. Make sure that the attachment integration component is last in sequence.
"Using Inline XML to Exchange Attachments" on page 222	New topic. You can exchange an entire business object, including any attachments, as a single XML file. Attachments are included within the body of the inline XML attachment.

What's New in Integration Platform Technologies: Siebel Enterprise Application Integration, Version 7.8, Rev. A

Table 2 lists changes described in this version of the documentation to support release 7.8 of the software.

Table 2. New Product Features in Integration Platform Technologies: Siebel Enterprise Application Integration, Version 7.8, Rev. A

Topic	Description
Chapter 6, "EAI Siebel Adapter"	Chapter was revised to provide clarification for using the EAI Siebel Adapter.
Chapter 10, "External Business Components"	Chapter was revised to provide clarification for using External Business Components, including adding a troubleshooting section.

What's New in Integration Platform Technologies: Siebel Enterprise Application Integration, Version 7.8

Table 3 lists changes described in this version of the documentation to support release 7.8 of the software.

Table 3. New Product Features in Integration Platform Technologies: Siebel Enterprise Application Integration, Version 7.8

Topic	Description
"About Siebel Authentication and Session Management SOAP Headers" on page 95	Session Management support added to Web Services. SOAP headers are being utilized for passing of the Session and Authentication information.
"About Web Services and Web Single Sign-On Authentication" on page 102	Web Single Sign-On support added to Web Services.
"Execute Method" on page 124	Two operations added to the Execute method of the EAI Siebel Adapter will explicitly skip processing parts of a message. The new operations provide hints to the EAI Siebel Adapter. The operations are useful when working with large and complex messages. The new operations are as follows: ■ **skipnode.** This will instruct the EAI Siebel Adapter to avoid setting field values for the particular node but will process children like normal. ■ **skiptree.** This will instruct the EAI Siebel Adapter to stop processing the tree rooted at this particular node.
Chapter 7, "EAI UI Data Adapter"	New chapter added to define the EAI UI Data Adapter business service.

2 Integration Objects

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

The chapter consists of the following topics:

Integration Objects Terminology

This chapter describes the concepts that are often referred to when using inconsistent terminology on different systems. Table 4 has been included to clarify the information in this chapter by providing a standard terminology for these concepts.

Table 4. Integration Objects Terminology

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

Table 4. Integration Objects Terminology

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

About Integration Objects

Siebel integration objects allow you to represent integration metadata for Siebel business objects, XML, SAP IDOCs, and SAP BAPIs as common structures that the EAI (Enterprise Application Integration) 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.

For more information, see *Business Processes and Rules: Siebel Enterprise Application Integration*.

NOTE: The Siebel Bookshelf is available on Oracle Technology Network (OTN) and Oracle E-Delivery. It might also be installed locally on your intranet or on a network location.

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

When planning your integration project, consider several issues:

- How much data transformation does your message require?

- At what point in the process do you perform the data transformation?
- Is a confirmation message response to the sender required?
- Are there data items in the originating data source that should not be replicated in the receiving data source, or that should replace existing data in the receiving data source?

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

Integration Object Base Object Type

Each integration object created in Siebel Tools has to be based on one of the base object types presented in Table 5. 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 5. Integration Object Base Object Types

Base Object Type	Description
None	For internal use only.
OLE DB	Used to expose Siebel business component as an OLE DB rowset that can be used by OLEDB consumers such as Excel, Word, and so on. The OLE DB Provider only accepts integration objects of this type.
SAP BAPI Input	Used to represent the input structure of an SAP RFC or BAPI function call. For information, see the *Siebel Connector for SAP R/3 Guide*.
SAP BAPI Output	Used to represent the output structure of an SAP RFC or BAPI function call. For information, see the *Siebel Connector for SAP R/3 Guide*.
SAP IDOC	Used with the IDOC Adapter and Receiver in version 6.x and 7.0. For information, see the *Siebel Connector for SAP R/3 Guide*.
SAP IDOC Adapter	Used to represent an SAP IDOC structure. For information, see the *Siebel Connector for SAP R/3 Guide*.
SQL	Used for manually creating integration objects. Only the EAI SQL Adapter accepts integration objects of this type.
SQL Database Wizard	Used by the Database Wizard for the integration object it creates. Only the EAI SQL Adapter accepts integration objects of this type.
SQL Oracle Wizard	Used by the Oracle Wizard for the integration object it creates. Only the EAI SQL Adapter accepts integration objects of this type.
Siebel Business Object	Used by the Integration Object Builder wizard for the integration object it creates. The EAI Siebel Adapter accepts only the integration object of this type.

Table 5. Integration Object Base Object Types

Base Object Type	Description
Table	Obsolete.
XML	Used to represent external XML Schema such as DTD or XSD. For information on DTD and XSD, see *XML Reference: Siebel Enterprise Application Integration*.

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.

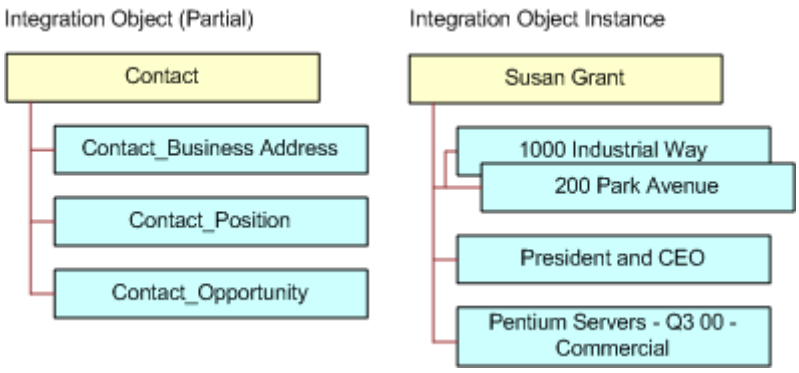


Figure 1. Integration Object and Integration Object Instance

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 Integration object Wizard and Generate Schema Wizard. The Generate Code wizard (not shown) works in the same manner as the Generate Schema wizard, but it generates Java classes.

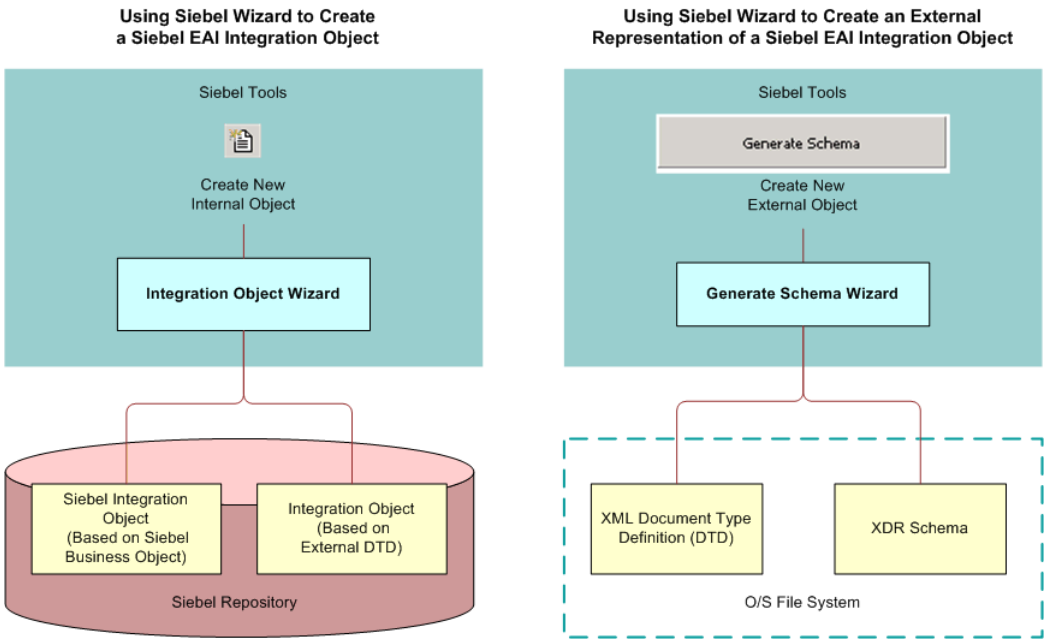


Figure 2. Integration Object Wizards

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). To access this wizard, navigate to the New Object Wizards dialog box in Siebel Tools, and 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. To access this wizard, navigate to the Integration Objects list in Siebel Tools and select an integration object. Then click Generate Schema to start the Generate XML Schema wizard.

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

NOTE: Specific instructions on how to use these wizards appear throughout the Siebel 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.

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

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

Each child component can have one or more child components. In Siebel Tools, if you look at the integration components for an integration object you have created, you see that each component can have one or more fields. Figure 3 on page 17 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 may choose to inactivate components and fields. By inactivating components and fields, you can define the structure of the integration object instances entering or leaving the system.

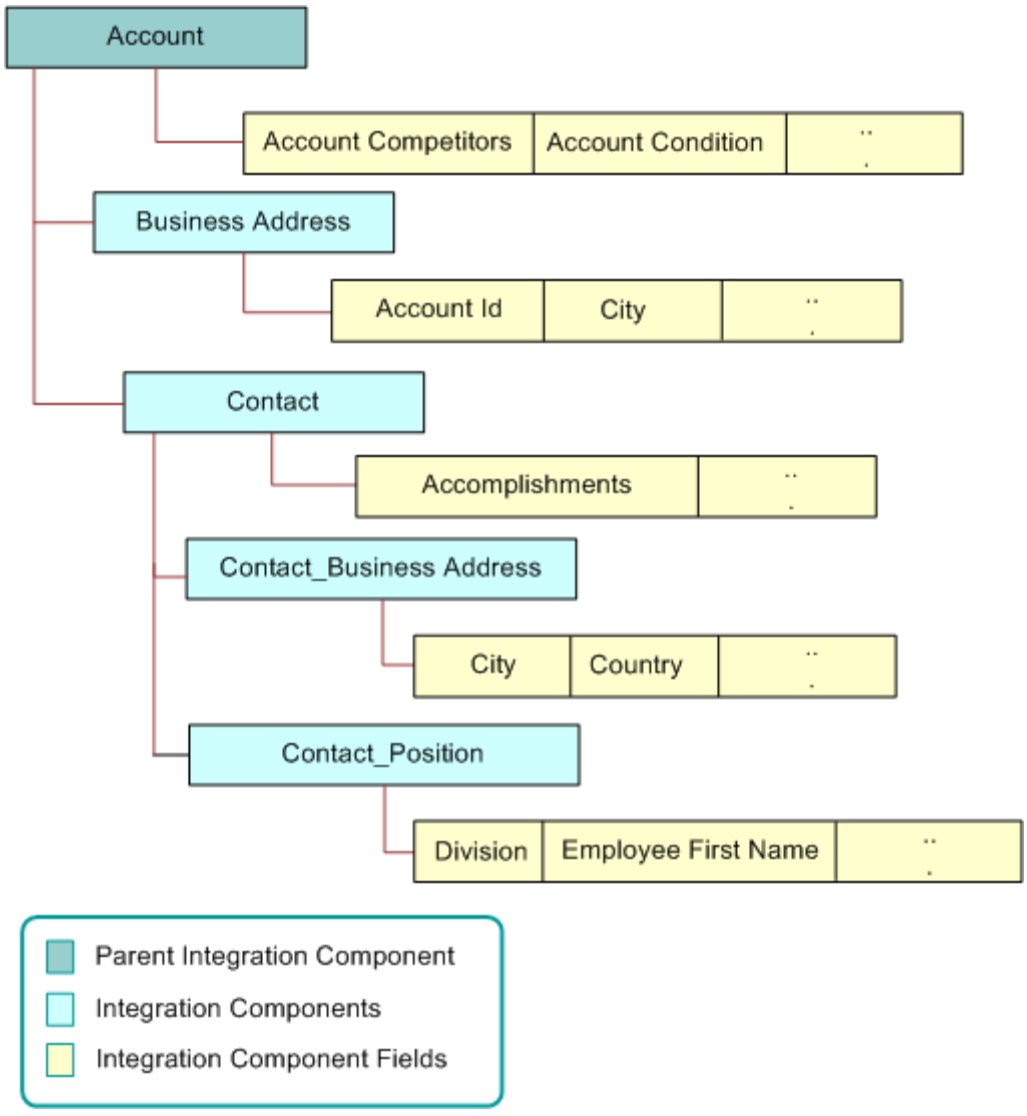


Figure 3. Representation of Partial Account Integration Object

About 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 multi-value groups (MVGs). User properties *Association* and *MVGAssociation* on the integration component denote that the corresponding business component is an associated business component or an associated MVG, respectively. For fields that are defined on MVG associations, *External Name* denotes the name of the business component field as it appears on the parent business component, and the user property *AssocFieldName* denotes the name of the business component field as it appears on the MVG business component.

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

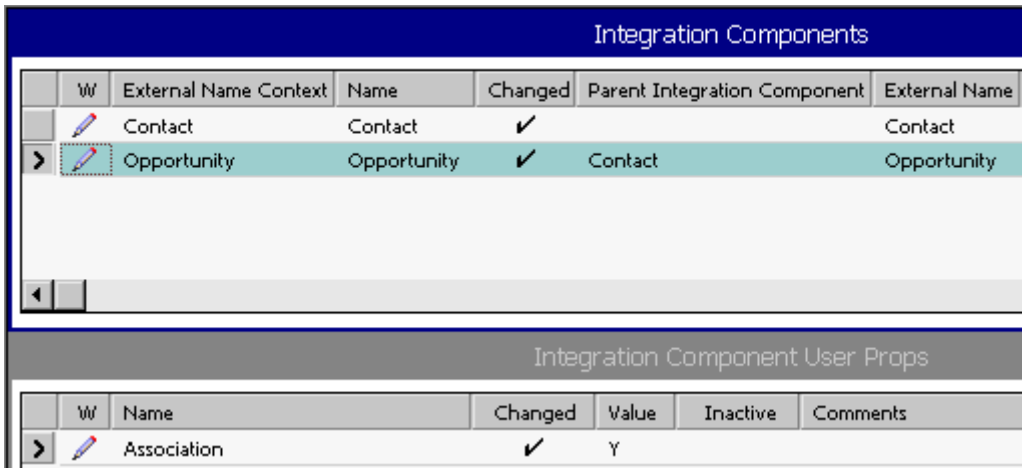


Figure 4. Integration Component Representation of Association

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

About 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 may require that only the integration component instance that corresponds to the primary MVG be part of the integration object instance. In relation to Account and Business Addresses this means that only the primary address should be part of the Account integration object instance. The primary address can be obtained by one of the following steps:

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

In Siebel Tools, if a Siebel business component contains an MVG, the MVG is represented in several screens as illustrated in the following sections.

Screen 1: Fields View for an MVG Field in a Business Component

For example, as illustrated in Figure 5, the Account business component contains a multi-value group field, the Address Id.

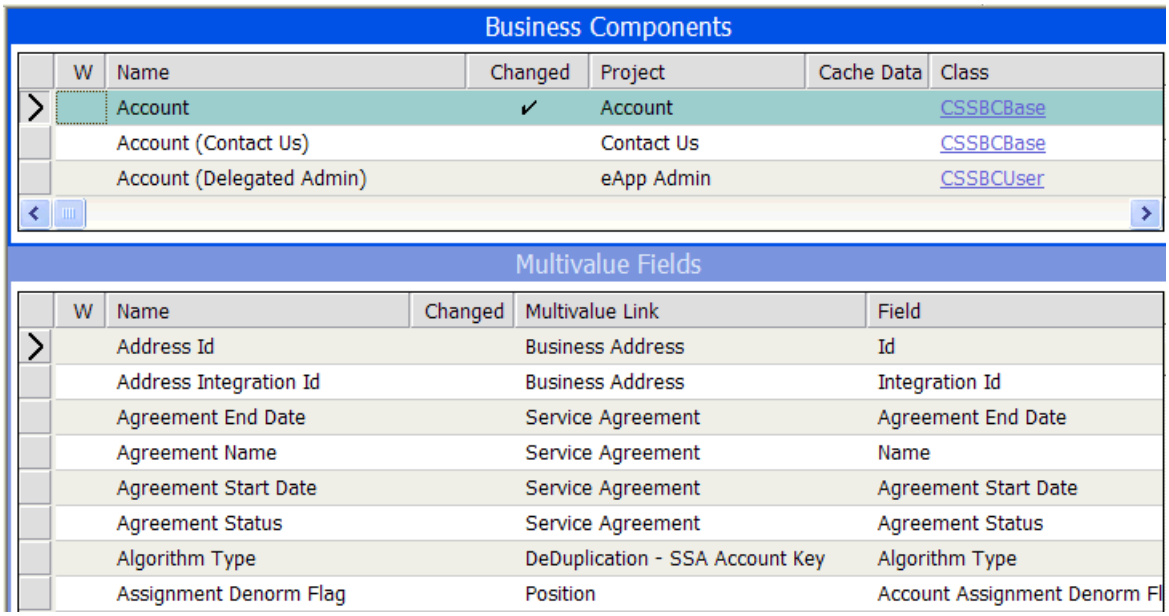


Figure 5. Address Id MVG Field in the Account Business Component

Screen 2: Multivalue Links in a Business Component

As shown in Figure 6, the multivalue link property has the value Business Address. If you navigate to the Multi Value Link screen, you see that the Business Address multivalue link has the value Business Address as its Destination Business Component.

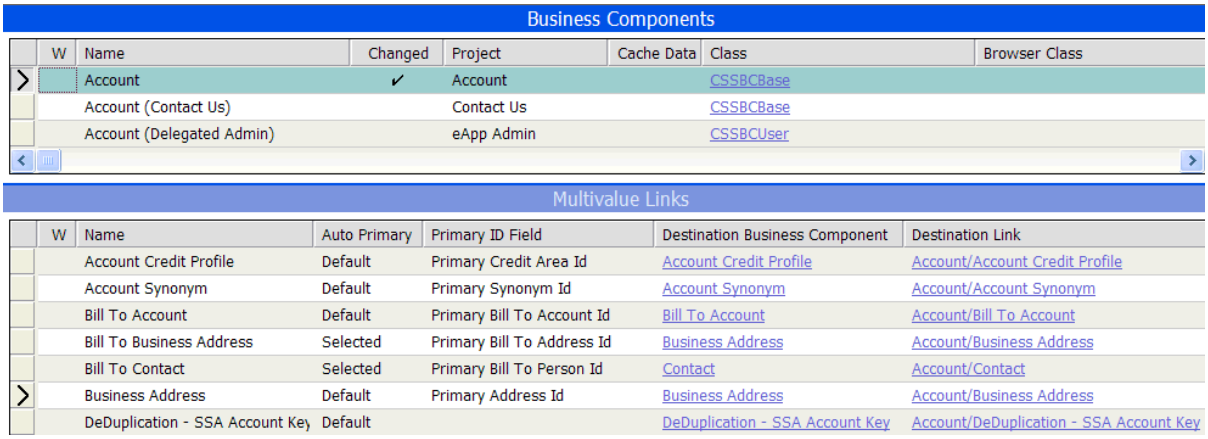


Figure 6. Destination Business Component

Screen 3: Fields View After Adding Multivalue Link

As shown in Figure 7, the Business Address multivalue link has Business Address as its Destination Business Component. This means that there is another business component named Business Address that contains the fields that are collectively represented by Address Id in the Account business component.

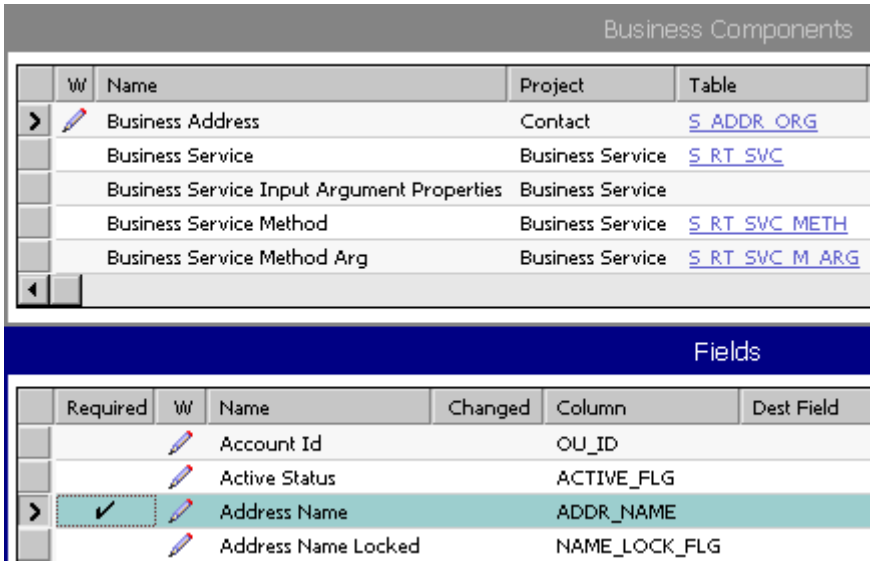


Figure 7. Business Address Business Component

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.

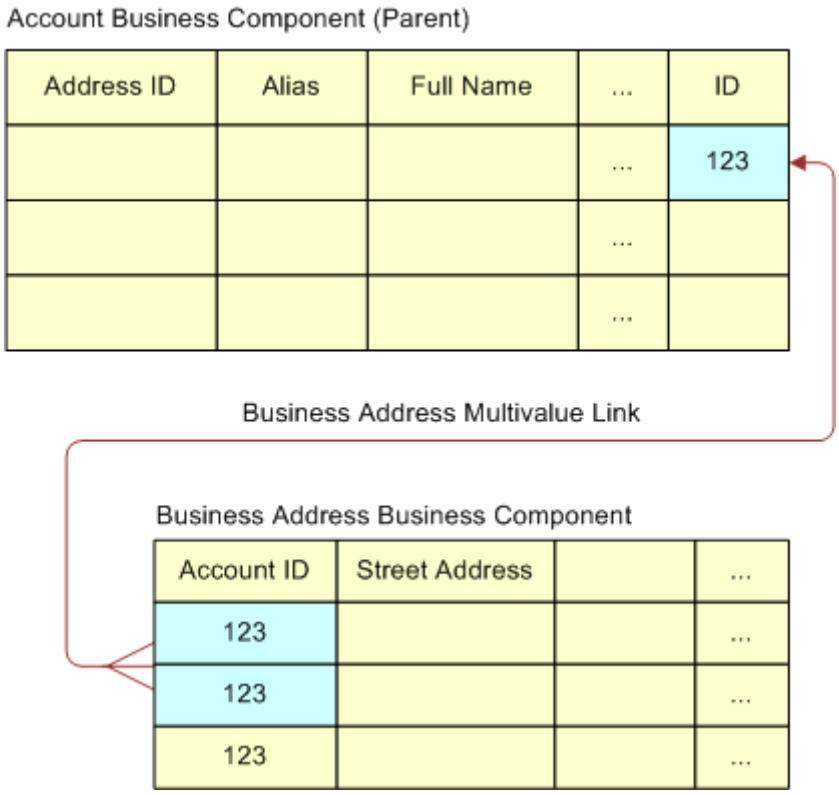


Figure 8. Address Id Field and Business Address MVG

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, 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.

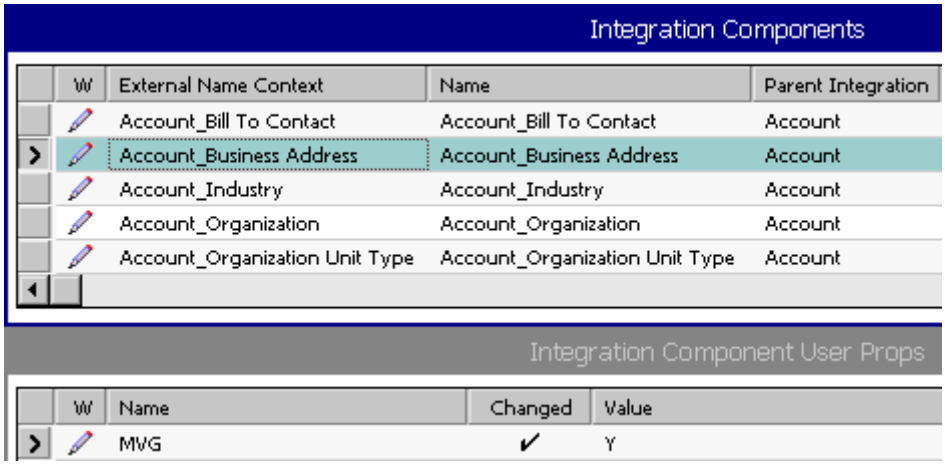


Figure 9. Integration Component Based on MVG Business Component

The EAI Siebel Adapter needs to know the names of the MVG fields as they are defined in the parent business component—in this example, Account—and also the names of the MVG fields as they are known in the business component that represents the MVG—in this example, Account Business Address. As shown in Figure 10, the integration component fields represent the MVG.

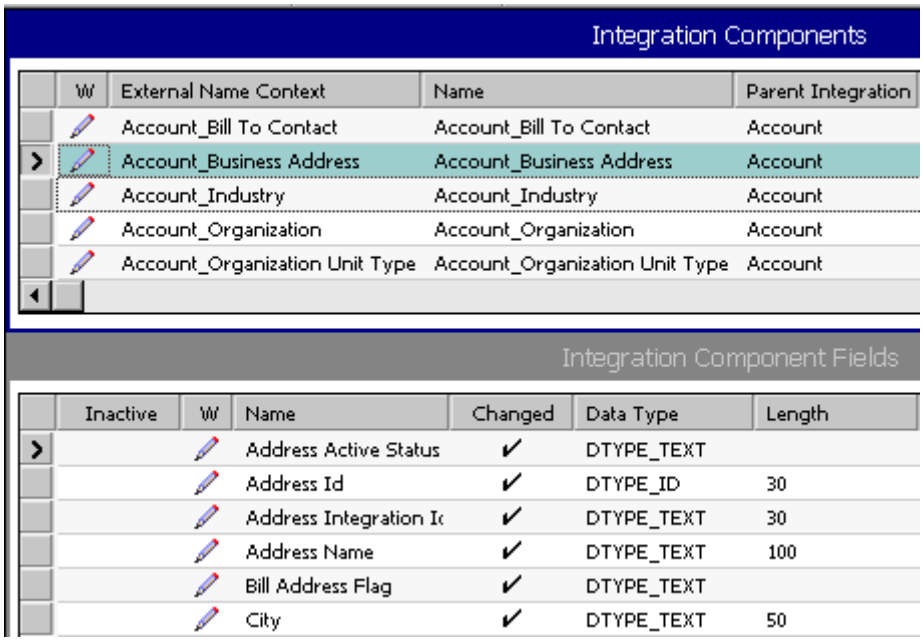


Figure 10. Integration Component Fields Representing MVG

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

About Validation of Integration Components 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, 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, 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).

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 needs to 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 example above: Account) is not defined, then any integration component fields that are mapped to columns in the U1 index of business component's base table, and are present in the pickmap will be used by the EAI Siebel Adapter to find the matching record in the picklist. (In the example above, this would be Account and Account Location.)

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

In order 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 and empty in the input property set.

About Calculated Fields and Integration Objects

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

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

About 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 may cause a query on that integration component to not find existing rows.

About Operation Controls for Integration Components

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

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

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

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

About 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 the value of each property should contain the name of the field that the associated field is dependent on. 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.

About 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 need to set primaries in addition to modifying the linked component, use both links and multivalue links in your integration object. It is recommended that the EAI Siebel Adapter use the multivalue link only after it processes the component through the link; therefore, the link or the Association component has a smaller external sequence number than the related MVG or MVGAssociation component. See “About the Structure of Integration Objects” on page 16 for an example.

About 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, 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 Keys

There are multiple types of integration component keys:

- **User Key.** See “About User Keys” on page 27.
- **Status Key.** See “About Status Keys” on page 32.
- **Hierarchy Parent Key.** See “About Using the Hierarchy Parent Key” on page 33.
- **Hierarchy Root Key.** See “About Using the Hierarchy Root Key” on page 33.
- **Modification Key.** See “Configuring the EAI Siebel Adapter for Concurrency Control” on page 144.
- **Foreign Key.** See the *Siebel Connector for Oracle Applications* guide.

- **Target Key.** See the *Siebel Connector for Oracle Applications* guide.

NOTE: There should be just one integration component key for every type of key except the user key. For example, if there are two Hierarchy Parent Keys defined for an integration component, the EAI Siebel Adapter picks the first one and ignores the second one.

About User Keys

User key is a group of fields whose values must uniquely identify a Siebel business component record. During inbound integration, user keys are used to determine whether the incoming data updates an existing record or inserts a new one. The Integration Object Builder wizard automatically creates some user keys based on characteristics discussed in "About User Key Generation Algorithm" on page 28. Make sure that the generated user keys match your business requirements; otherwise, inactivate them or add new user keys as appropriate.

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

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

A sequence of integration component user keys is defined on each integration component definition, each of which contains a set of fields. During processing of integration component instance, 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 Account Name and Account Integration Id field present. When the EAI Siebel Adapter performs validation, it first checks the Account Name and Account Location field (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 Account Location field present, but set to null, then the EAI Siebel Adapter picks 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 case of the user key fields do not match. To avoid this, use the Force Case property on the business component field to make sure that user key fields are always stored in one case.

About User Key Generation Algorithm

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

To understand how the Integration Object Builder wizard determines valid integration component keys, you can simulate the process of validating the user keys. For example, determine the table on which your business component is based. In Siebel Tools, you can look up this information yourself. Navigate to the Business Components screen, and select a business component and check the Table column.

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

For example, as shown in Figure 11, the table S_CONTACT has several user keys.

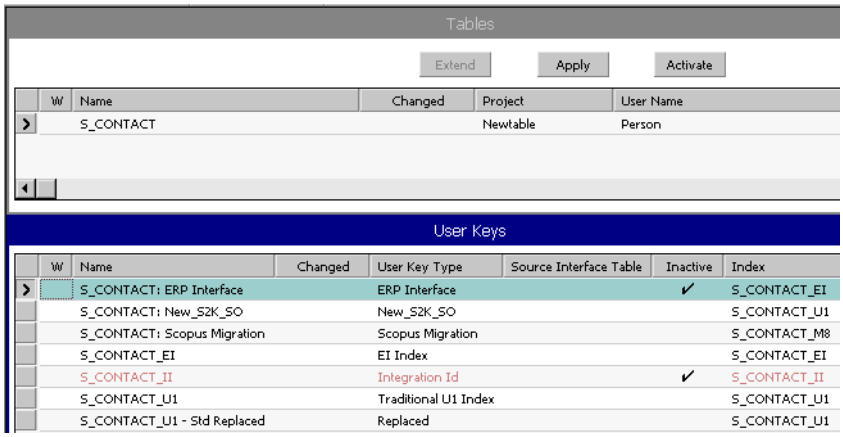


Figure 11. User Keys for Table S_CONTACT

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

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

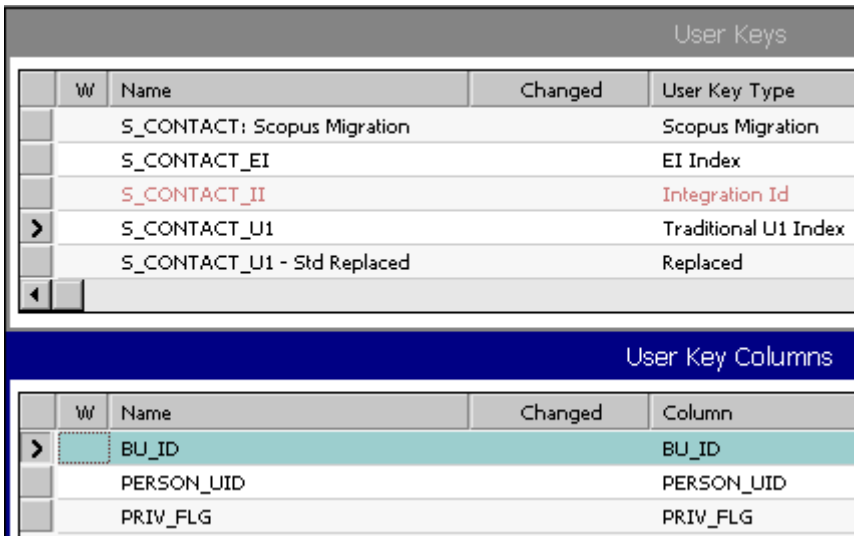


Figure 12. User Key Columns for the S_CONTACT_U1 User Key

If the columns of the user key are exposed in the business component, and those columns are not foreign keys, the Integration Object Builder wizard creates an integration component key based on the table's user key. The Integration Object Builder wizard also defines one integration component key field corresponding to each of the table's user key columns. For example, in Figure 13, the user key columns are exposed in the Integration Component Fields applet for the Contact integration component.

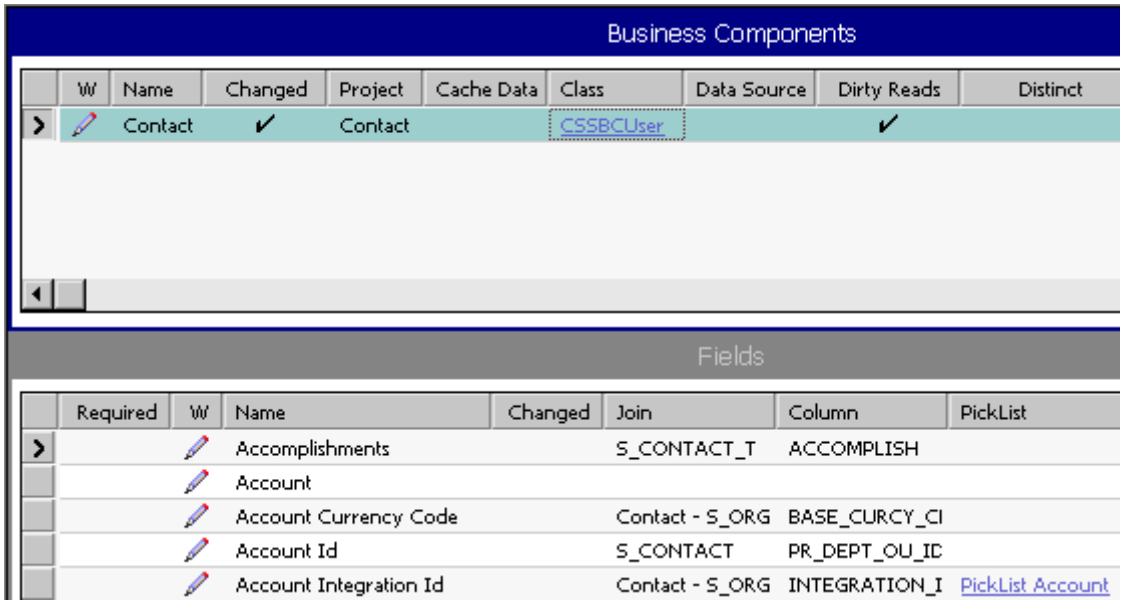


Figure 13. Integration Component Field List

The Integration Object Builder wizard, for the preceding example, builds the integration component keys based on these table user keys. As illustrated in Figure 14, the wizard defines one integration component key field for each table user key column.

Integration Components

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

Integration Component Keys

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

Figure 14. Integration Component Keys for Each Table User Key Column

Each valid integration component key contains fields. For example, as shown in Figure 15, for the Contact integration component, User Key 3 is made up of five fields: CSN, First Name, Last Name, Middle Name, and Personal Contact.

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

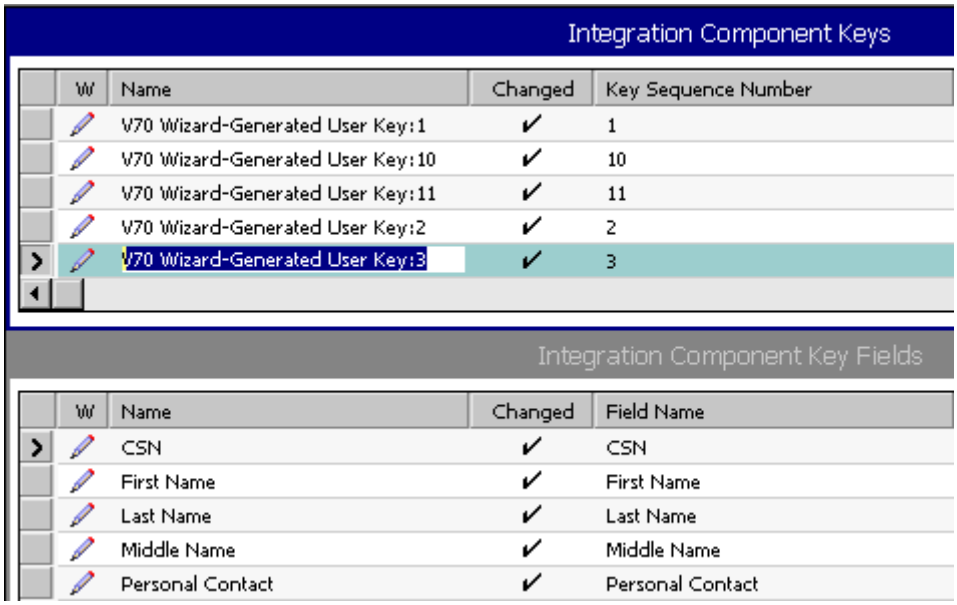


Figure 15. Contact Integration Component Key Fields

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

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

About Status Keys

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

For many integrations, you want to know the status. For example, if you are sending an order request you want to know the ID of the Order created so that you can query on the order in the future. You can set the Status Object of the EAI Siebel Adapter to True to return an integration object instance as a status object.

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

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

About Using the Hierarchy Parent Key

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

For example, when you insert quotes, each quote item in turn can have more quote items. In this case, the 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).

About Using the Hierarchy Root Key

The Hierarchy Root Key is an optional key that is useful only when integration objects have a homogeneous hierarchy. You can use this key to improve performance. The Hierarchy Root Key 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.

Permission Flags for Integration Components

Each business component, link, MVG, and integration object user property has settings such as No Update, No Delete, and No Insert. These settings indicate the type of operations that cannot be performed on that object. In order for 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, a warning message is logged in the log file and processing continues. Otherwise, an error message is returned, and the transaction is rolled back. You set the permission settings by using user properties on the integration object, component, or field.

Table 6 illustrates which permissions influence which operation type on an integration component.

Table 6. Permission Flags for an Integration Component

Permission Layer	Checked by...	Integration Component Type		
		Standard	MVG	Association
Integration Object Component	EAI Siebel Adapter	Yes	Yes	Yes
Integration Component		Yes	Yes	Yes
Integration Field (Update Only)		Yes	Yes	Yes
Link	Object Manager	Yes	Yes	Yes
Multivalue Link (MVL)		No	Yes	No
Business Component (Overridden by AdminMode)		Yes	Yes	Yes
Business Component Field		Yes	Yes	Yes

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

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, 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 need to expose outside of Siebel Business Applications through HTTP inbound or MQSeries Receiver server components. If this user property is not specified, 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 consists of the following topics:

About the Integration Object Builder

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

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

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

- Create integration objects using the EAI Siebel Wizard.

  For information, see "Creating Integration Objects Using the EAI Siebel Wizard" on page 38.

- Fine-tune your integration object.

  For information, see "Siebel Integration Object Fine-Tuning" on page 40.

- Validate your integration object.

  For information, see "Validating Integration Objects" on page 41.

Creating Integration Objects Using the EAI Siebel Wizard

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

To create a new Siebel integration object

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

      NOTE: The name of an integration object must be unique among other integration objects. 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.

7 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, the children below that component are deselected as well. You cannot include a child component without also including the parent. The Integration Object Builder enforces this rule by automatically selecting the parent of any child you choose to include.

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

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

b Select the Contact component. When selecting a child component, its parent component is also selected, but none of the components below the child component are selected. You must individually select the ones you want.

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

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

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

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

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

NOTE: 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.

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

The Integration Component Fields applet displays the list of fields for that component. Note the system fields Conflict Id, Created, Id, Mod Id, Updated, operation, and searchspec in the list. This setting prevents 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" on page 138.

Creating an Integration Object Based on Another Root Business Component

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

To create an integration object based on another root business component

1. Modify the business object definition in Siebel Tools to have that particular root business component as the Primary Business Component.
2. Run the Integration Object Builder wizard and choose the business object you want to use.
3. Undo the changes to the business object definition.

Step 3 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 it does not affect any business logic.

Siebel Integration Object Fine-Tuning

After you create your integration object you must fine-tune and customize your integration object based on your business requirements. The following is a list of the most common practices in fine-tuning an integration object:

- 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. For information on the implications of activating such fields, see “About Calculated Fields and Integration Objects” on page 24 and “About Inner Joins and Integration Components” on page 25.
- 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 51.

Validating Integration Objects

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

To validate your integration object

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

NOTE: The integration objects you create in Siebel Tools must be compiled into the Siebel.srf file. When you test the integration object, you must copy the compiled SRF to your *SIEBSRVR_ROOT*\OBJECTS directory.

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

Synchronizing Integration Objects

To help simplify the synchronization task, Siebel EAI provides an integration object synchronization utility. Although the process of synchronizing your integration object with its underlying business object is straightforward, 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.

The following checklist gives the high-level procedures for updating an integration object:

- Run the Synchronization wizard.

  For information, see "Updating the Entire Integration Object" on page 44.

- Modify the newly updated integration object as needed, using the DIFF tool and a copy of the original integration object as reference.

  For information, see *Using Siebel Tools*.

- Run Validation.

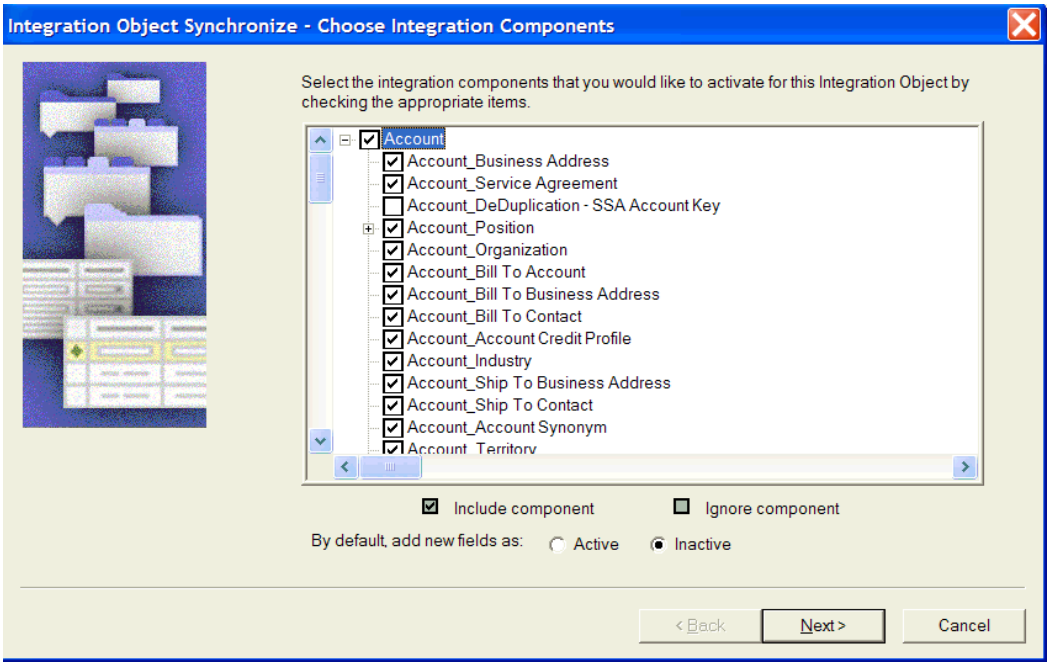
  For information, see "Validating Integration Objects" on page 41.

To update an integration object with new and current business object definitions

1. Access the integration object you want to update in Siebel Tools.
2. Run the Synchronization wizard by clicking Synchronize in the Integration Objects list applet.

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

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



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

   b. Uncheck the boxes beside the objects and components you do not want to include in the synchronization of your Siebel integration object. Note that only the objects that are included in the new integration object are marked.

c 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.

d 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.

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

e If no changes are needed, 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 44 and "About Deleting a Component from the Integration Object" on page 46.

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

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

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

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

Synchronization Rules

During the synchronization process, the Synchronization wizard follows particular update rules. Consider a simple example involving the Siebel Account integration object with only Contact and its child components marked as active in the object. Figure 16 helps you to visualize this example.

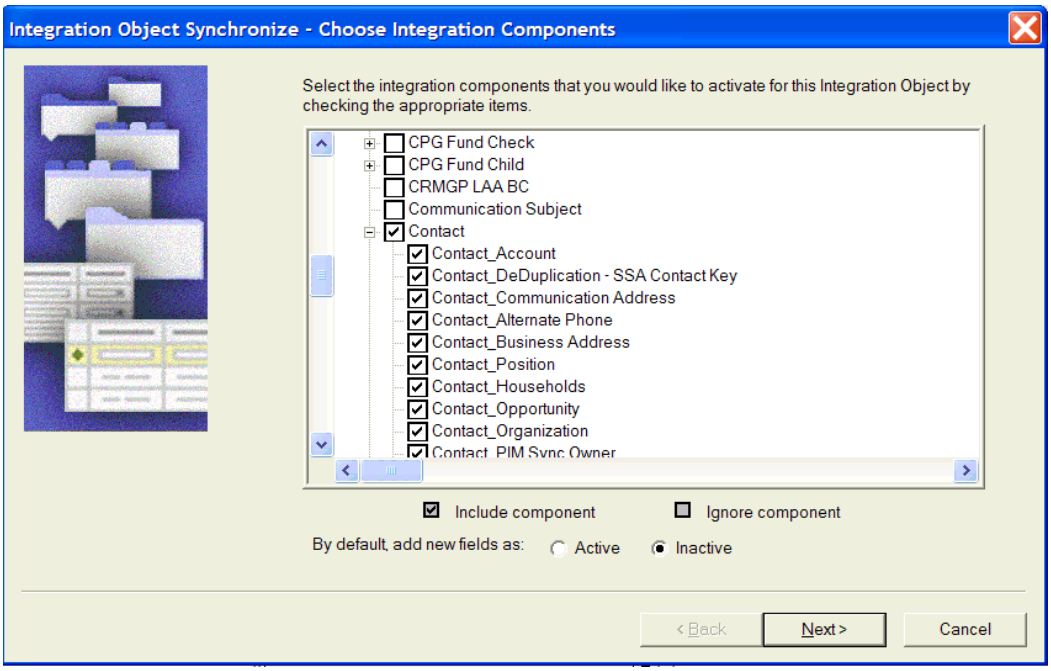


Figure 16. Example of Selected Integration Components

Because the Account component is the parent of Contact, it is also selected, even though you cannot see it in Figure 16.

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 17 illustrates how the Synchronization wizard takes into account any changes.

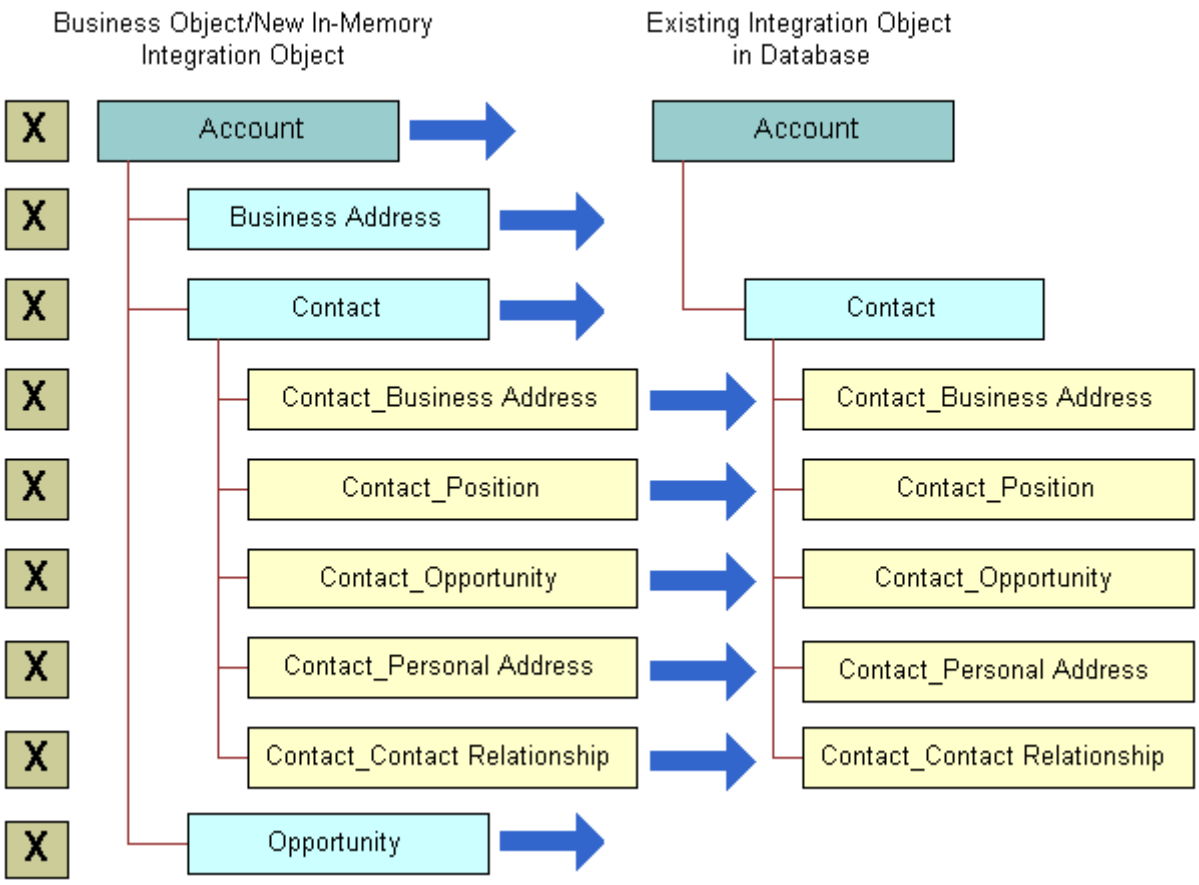


Figure 17. Synchronizing the Integration Object

Figure 18 shows how the resulting integration object is structured after the synchronization.

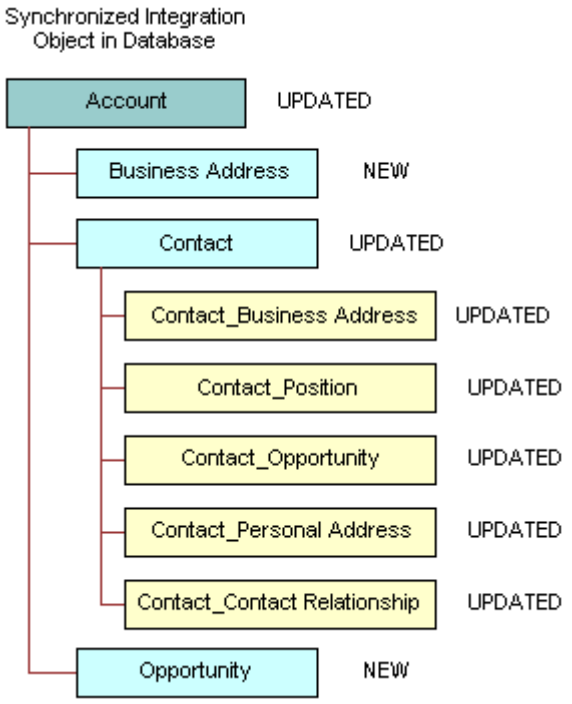


Figure 18. Completely Updated Integration Object

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.

About Deleting a Component from the Integration Object

If you choose to deselect a component in the Synchronization wizard, 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 19 on page 47, 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 19 illustrates this concept.

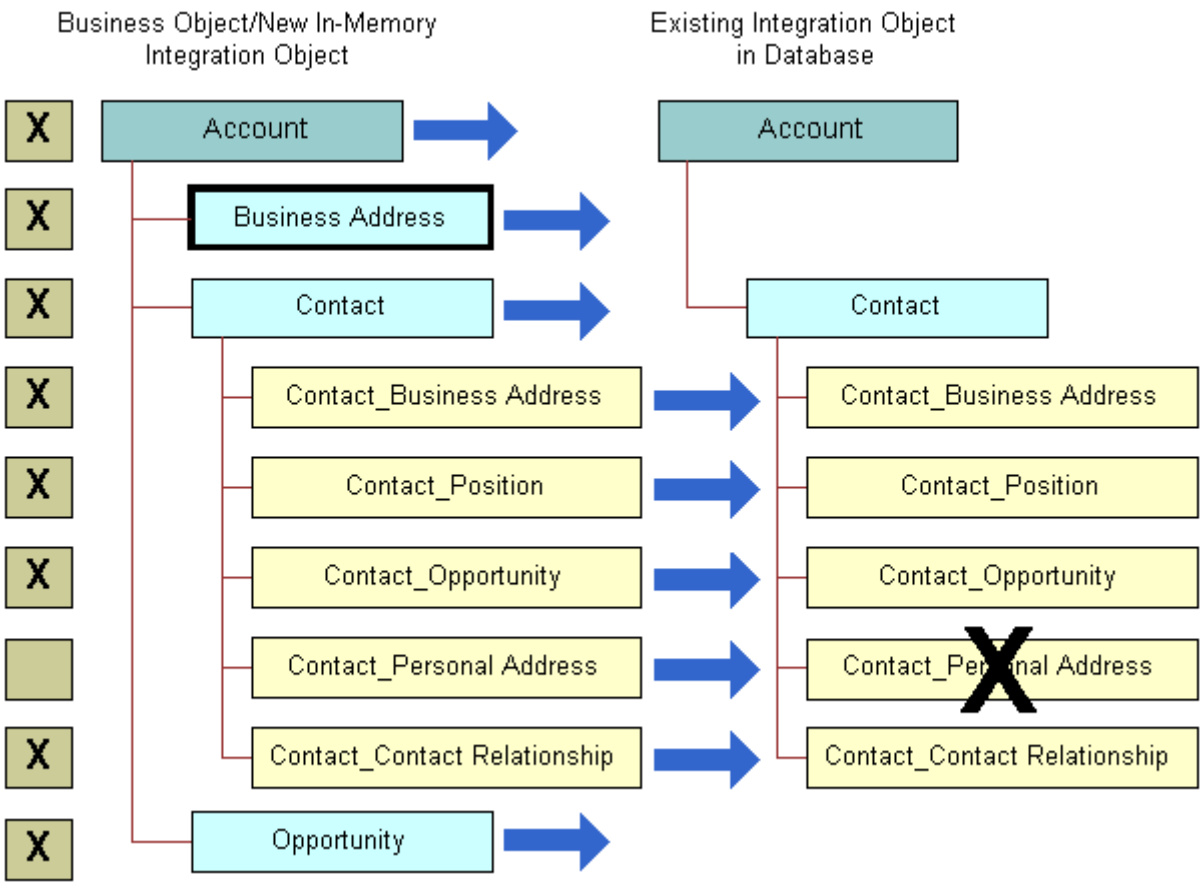


Figure 19. Deleting a Component from the Integration Object

Figure 20 shows the integration object after synchronization.

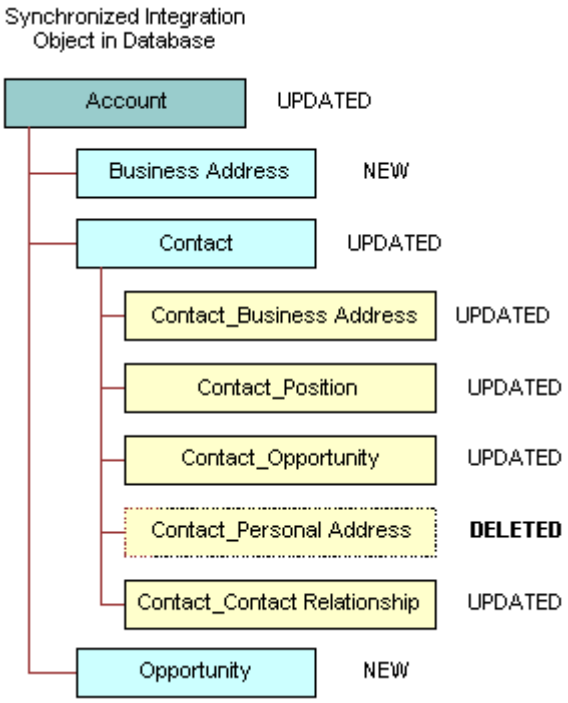


Figure 20. Synchronization Resulting in a Deleted Component

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, deleting a component from the integration object method accomplishes that goal.

As the examples illustrate, you must be aware of the possible changes that can occur when you are synchronizing business objects and integration objects. The Synchronization wizard can provide assistance in managing your integration objects, but you 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.

About the EAI Siebel Wizard

You can use the EAI Siebel Wizard to create integration objects that represent Siebel business objects. During the process of creating a new integration object, described in "About the Integration Object Builder" on page 37, you can choose the EAI Siebel Wizard as the business service to help create the object. This wizard understands the structure of Siebel business objects. As shown in Figure 21, the wizard returns a list of the available business objects from which you can choose one to base your integration object on.

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

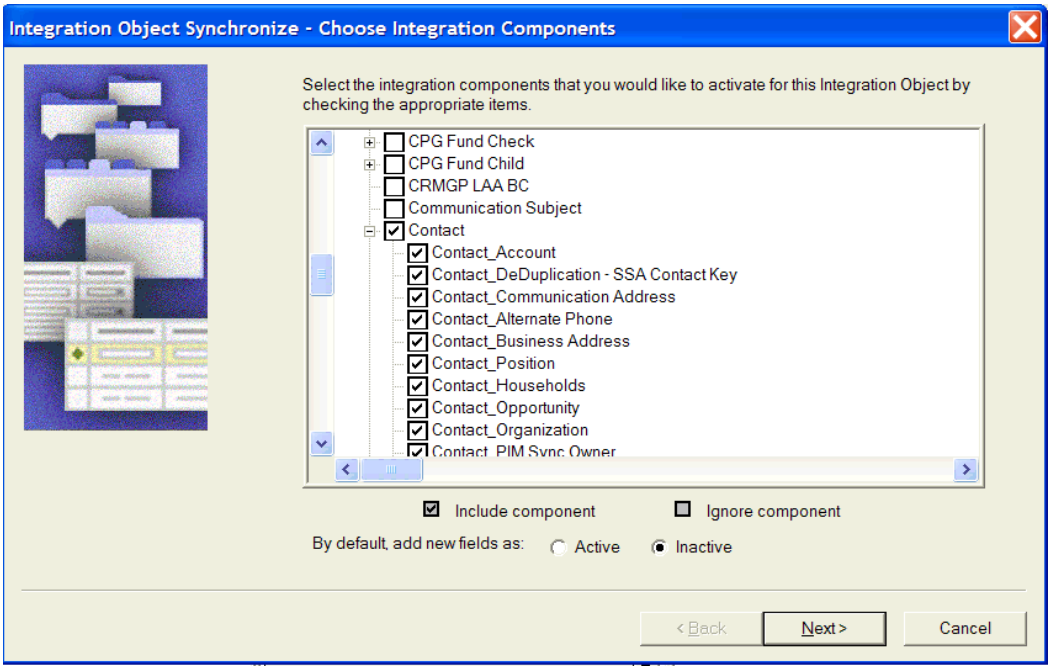


Figure 21. Activated Components in the Contact Integration Object

After the wizard creates your integration object, you can edit the object in Siebel Tools, as shown in Figure 22. You might choose to drill down into the integration components and activate or inactivate particular components or even particular fields within one or more components.

NOTE: Always inactivate the fields rather than delete them, even though the net effect will be the same. When you execute the synchronization task, using the Siebel EAI sync utility in Siebel Tools, inactivated fields remain inactive, while the deleted fields are created as active fields in the integration object.

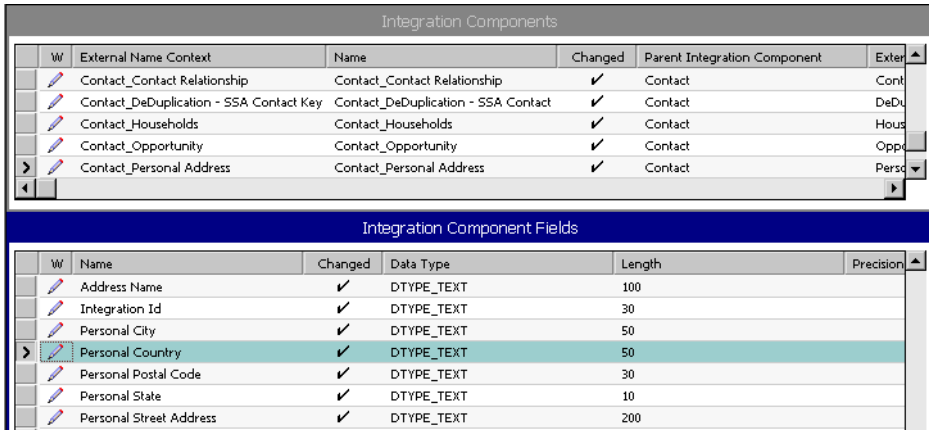


Figure 22. Activated Components in the Contact Integration Object

Best Practices for Maintaining Siebel Integration Objects

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

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

- Name any user key that you add, which 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.

Resolving Synchronization Conflicts for Integration Objects and User Properties

This section serves as a guide to resolving synchronization conflicts should 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

Integration Object Metadata	Merging Rules
Objects	Validate that Business Object still exists.
Components	Present the tree of components based on current business object definition. The components present in the current integration object are checked in the UI tree, other components presented as Inactive. User decides which components to add or delete. This is done by the Synchronization wizard UI.
Fields	Keep the current integration component fields if still present in the business component, otherwise delete. Add new fields in a way that does not conflict with existing ones (see 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, delete it. Active/Inactive—Preserve the current integration component field value unless Business Component Field is Required (field must be present during Insert). Otherwise, new fields are created Inactive.
XML Properties	Preserve the current integration object values to keep XML compatible. Add new components/fields properties avoiding conflict with existing XML. XML Properties are processed according to the XML sequence. New components/fields that sequence within the parent component element will be higher than current. Existing processing code should be reused (and checked for correct behavior).

Table 7. Merging Logic Used for Synchronizing Integration Objects

Integration Object Metadata	Merging Rules
External Sequence on components or fields	Preserve the component or field sequence within the parent component. Set the sequence on new components or fields higher than the existing ones.
Name	Preserve Names in the current integration object.

User key, Hierarchy key, Other keys (for example, Status Key)	Existing Keys:
	■ Keep existing keys as Active if all the key fields are Active.
	■ Keep existing keys Inactive if Inactive already or make Inactive if any of the fields are Inactive.
	■ If a field is Inactive in an integration component, make it Inactive in the key. Make the key Inactive.
	■ If a field is not present in an integration component, delete it from the key. Make the key Inactive.
	New Keys:
	■ Create new keys as Inactive.
	■ If any of the key fields are Inactive, either:
	  ■ Do not create the key.
	  ■ Make fields Active in the integration component.

User Properties	Preserve valid cases, remove invalid ones, and generate warnings.

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

Table 8. Logic Used for Synchronizing User Properties

User Property Name	Values (Default is in *italics*)	Level (*O*bject, *C*omponent, or *F*ield)	Merging Rules
Association	Y,*N*	C	Siebel Wizard generates the value based on current business component definition. The Wizard overwrites the user change, because in order for the integration component to be functional, the User Property has to be consistent with the business component.
MVG	Y,*N*	C	Siebel Wizard generates the value based on the current business component definition. The Wizard overwrites the user change, because in order for integration component to be functional, the User Property has to be consistent with the business component. IsPrimaryMVG system field is created in the merged integration object.
Picklist	Y,*N*	F	Siebel Wizard generated. The user change is kept if valid (if Picklist component). Review the input object for a user property of PICKLIST. Copy from the current field.
PicklistUserKeys	Any active fields.	F	Entered by user, keep only Active fields. User Property is valid only if PICKLIST is set to Y on the integration component. If no Active fields left, remove the user property.

Table 8. Logic Used for Synchronizing User Properties

User Property Name	Values (Default is in *italics*)	Level (*O*bject, *C*omponent, or *F*ield)	Merging Rules
IgnoreBoundedPicklist	Y,*N*	O, C, F	Entered by user, keep if valid (if component Picklist is set to Y).
MVGAssociation	Y,*N*	C	Siebel Wizard generates the value based on the current business component definition. The Wizard overwrites the user change, because in order for integration component to be functional, the User Property has to be consistent with the business component. IsPrimaryMVG system field is created in merged integration object.
MVGFieldName	Any valid field name in the MVG business component.	F	Siebel Wizard generates the value based on current business component definition. The Wizard overwrites the user change, because in order for integration component to be functional, the User Property has to be consistent with the business component. (component MVG is set to Y)
AssocFieldName	Any valid field name in the Association business component.	F	Siebel Wizard generates the value based on current business component definition. The Wizard overwrites the user change, because in order for the integration component to be functional, the User Property has to be consistent with the business component. (component MVGAssociation is set to Y)

Table 8. Logic Used for Synchronizing User Properties

User Property Name	Values (Default is in *italics*)	Level (*O*bject, *C*omponent, or *F*ield)	Merging Rules
NoInsert, NoDelete, NoUpdate, NoQuery, NoSynchronize	Y,*N*	C, F (NoUpdate)	Entered by the user. Keep the current value.
FieldDependency*FieldName*	Any active integration component name within the same integration component.	F	Entered by the user. Keep the current value if valid (if *FieldName* field is Active).
AdminMode	Y, *N*	C, O	Entered by the user; if the value exists, keep it. Otherwise, the wizard sets the value to N.
ViewMode	*All*, Manager, Sales Represent, and any others	O	Entered by the user; if the value exists, keep it. Otherwise, the wizard sets the value to All.
AllLangIndependentVals	Y,*N*	O	Entered by the user; if the value exists, keep it. Otherwise, the wizard sets the value to N.
IgnorePermissionErrorsOnUpdate, IgnorePermissionErrorsOnInsert, IgnorePermissionErrorsOnDelete	Y,*N*	C	Entered by the user. Keep the current value.
ForceUpdate	Y,*N*	O	Entered by the user. Keep current the value.
SupressQueryOnInsert	Y,*N*	C	Entered by the user. Keep the current value.

Example of 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 a Contact business object with Contact and Opportunity business components.

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

1. Start Siebel Tools.
2. Create a new project and lock it, or lock an existing project in which you want to create your integration object.
3. Choose File > New Object... to display the New Object Wizards dialog box.
4. Select the EAI tab and double-click the Integration Object icon.
5. In the Integration Object Builder wizard:
   a. Select the project you locked in Step 2.
   b. Select the EAI Siebel Wizard.
6. Click Next and in the second page of the Integration Object Builder wizard:
   a. Select the source object Contact to be the base for the new Siebel integration object.
   b. Type a unique name in the field for the new Siebel integration object, and click Next, for example, Sample Contact M:M.
   c. Select the source root for the new integration object from the list.
7. From the list of components, select Contact and Opportunity. There is also a component named Contact_Opportunity in the list. This component is an MVGAssociation component, and you pick it only if you need this integration object to set the primary opportunity for contact. For information on MVG, see "About MVGs in the EAI Siebel Adapter" on page 137.
8. 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.)
9. 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.)
10. Compile a new SRF file and copy the SRF file to the *SIEBSRVR_ROOT*/OBJECTS directory.

To test the newly created integration object

1. Start the Siebel client connected to Sample database.
2. Copy and modify the Import Account (File) and the Export Account (File) sample workflow processes to work with the Contact business object, instead of the Account business object.
3. Modify the Export Account (File) workflow process to invoke the EAI Siebel Adapter against the Sample Contact M:M integration object that you created, see "To create an integration object with a many-to-many business component" on page 56.
4. Run the workflow processes using the Workflow Process Simulator.

Generating Integration Object Schemas

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

To generate an integration object schema

1. In Siebel Tools, click on an integration object to make it the active object.
2. Click Generate Schema to access the Generate XML Schema wizard.
3. Choose the EAI XML DTD Generator business service to use as an example.

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

About Optimizing Performance for Using Integration Objects

To optimize your integration object performance, you may want to consider the following.

Size of Integration Object

The size of an integration object and its underlying business components can have an impact on the latency of 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 may 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, also causes fields be auto-filled. In this instance, providing a single value to a particular field causes the value of the field to be auto-filled. This occurs especially when the picklist is based on an MLOV. If the EAI Siebel Adapter is performing 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.

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.

Business Component Restrictions for Integration Components

The business components underlying the Integration Components may 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, 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, the setting will be ignored and a warning message will be logged.

Best Practices for Using Integration Components

The following are the best practices for using integration components:

- Familiarize yourself with the business logic in the business components. Integration designers should use the presentation layer, or the user interface to get a good sense of how the business component behaves, and what operations are allowed and not allowed.
- Design with performance in mind. For more information on performance and using integration objects, see "About Optimizing Performance for Using Integration Objects" on page 57.
- Design with maintenance in mind. For more information on maintenance, see "Best Practices for Maintaining Siebel Integration Objects" on page 50.
- Resolve configuration conflicts. During the development of your integration points, you might encounter issues with the configuration of business components that are configured to support interactive GUI usage, but do not satisfy your integration requirements.

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

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

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

  Scenario 2. A business component such as Internal Product is made read-only for regular GUI usage, but you want your integration process to be able to update the Internal Product business component.

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

  Scenario 3. Similar to scenario 2, a business component such as Internal Product is made read-only for regular GUI usage, but you want your integration process to be able to update the Internal Product business component. The only difference in this scenario is that the business component is used through a link that has NoUpdate property set to Y.

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

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

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

4 Business Services

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

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.

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 Siebel documentation. The use of undocumented services is not supported and can lead to undesired and unpredictable results.

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

- **In Siebel Tools.** Created at design time in Siebel Tools using Siebel VB or Siebel eScript. Design-time business services are stored in the Siebel repository (.SRF), so you have to compile the repository before testing them. 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 64.
- **In Siebel Client.** Created at run time in the Siebel Client using the Business Service Administration screens. Run-time business services are stored in the Siebel Database, so they can be tested right away. The run-time business services have to be manually moved over after an upgrade process. For information, see "Creating a Business Service in the Siebel Client" on page 67.

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

Business Service Structure

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

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

NOTE: For more information on business service methods and method arguments, see *Using Siebel Tools*.

About Property Sets

Property sets are used internally to represent Siebel EAI data. A property set is a logical memory structure that is used to pass the data between business services. Figure 23 illustrates the concept of a property set.

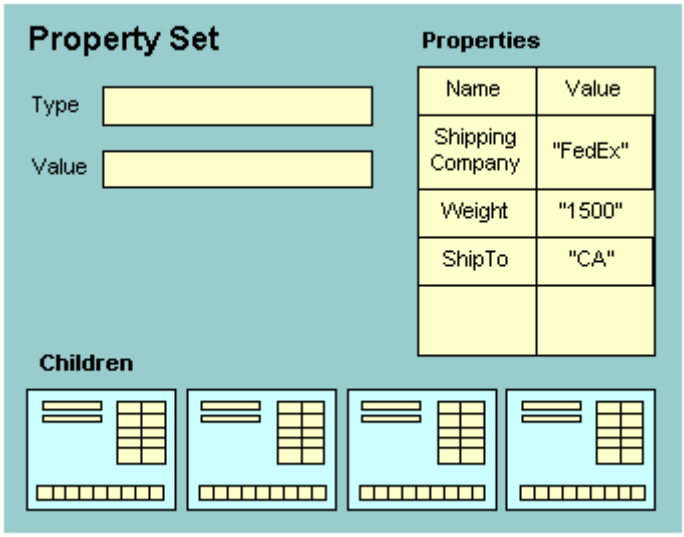


Figure 23. Property Set Structure

The property set consists of four parts:

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

  NOTE: In Siebel Tools, a Value argument to a method is shown with the name of <Value>, including the 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 may contain an Account with some set of contact records from the database. Each contact record is represented as a child property set.

NOTE: For information on property sets and their methods, see *Using Siebel Tools*.

Creating Business Services in Siebel Tools

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

- Define the Business Service

  For information, see "Defining a Business Service in Siebel Tools" on page 64.

- Define the Business Service Methods

  For information, see "Defining Business Service Methods" on page 65.

- Define the Business Service Methods Arguments

  For information, see "Defining Business Service Method Arguments" on page 65.

- Define Business Service Scripts

  For information, see "Defining and Writing Business Service Scripts" on page 66.

- Define Business Service User Properties

  For information, see "Defining Business Service User Properties" on page 66.

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

Defining a Business Service in Siebel Tools

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

To define a business service in Siebel Tools

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

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

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

   The list of predefined business services appears in the right panel.

4. Choose Edit > New Record to create a new business service.
5. Type a name for your business service in the Name field.
6. Type the name of the project you locked in Step 2, in the Project field.
7. Choose the appropriate class for your business service, from the Class picklist:

- Data transformation business services must use the CSSEAIDTEScriptService class.
- Other business services will typically use the CSSService class.

8 Step off the current record to save your changes.

Defining Business Service Methods

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

NOTE: For information on business service methods, see *Using Siebel Tools*.

To define a business service method

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

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

2 Choose Edit > New Record to create a new method.

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

Defining Business Service Method Arguments

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

To define the business service method arguments

1 With your business service selected, double-click the Business Service Method Arg folder, in the Tools Object Explorer, to display the Business Service Method Args list.

2 Choose Edit > New Record to create a blank method argument record.

3 Type the name of the argument in the Name field.

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

4 Enter the data type in the Data Type field.

5 Check the Optional check box if you do not want the argument to be required for the method.

6 Choose a Type for the argument. Refer to the following table for a list of different types and their descriptions.

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

Defining and Writing Business Service Scripts

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

To define and write the business service script

1 Start Siebel Tools.

2 Select the business service for which you want to write a script.

3 Right-click to display a pop-up menu.

4 Choose Edit Server Scripts.

5 Select either Siebel eScript or Visual Basic for your scripting language.

Service_PreInvokedMethod is selected as the event handler.

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

6 Type your script into the Script Editor.

NOTE: 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, also known as User Props, are optional variables that you can use to define default values for your business services. When a script or control invokes your business service, one of the first tasks the service performs is to check the user properties to gather any default values that will become input arguments to the service's methods.

To define business service user properties

1 With your business service selected, double-click the Business Service User Prop folder in the Tools Object Explorer to display the list of Business Service User Props.

2 Choose Edit > New Record to create a blank user property record.

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

4 Type a value in the Value field.

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

Creating a Business Service in the Siebel Client

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

To define a business service in the Siebel Client

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, your deployment platform must support Siebel VB.

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

Business Service Export and Import

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

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

Testing Your Business Service

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

To run the Business Service Simulator

1. Navigate to the Administration - Business Service > Simulator view.

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

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

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

   The result appears in the Output Property Set applet.

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

```
var svcReadFile = TheApplication().GetService("EAI XML Read from File") ;

var svcSaveData = TheApplication().GetService("EAI Siebel Adapter");

var child = TheApplication().NewPropertySet();

var psInputs = TheApplication().NewPropertySet();

var psOutputs = TheApplication().NewPropertySet();

var psOutputs2 = TheApplication().NewPropertySet();

var svcSaveData = TheApplication().GetService("EAI Siebel Adapter");

psInputs.SetProperty("FileName", "c:\\NewAccount.xml");

psOutputs.SetType "SiebelMessage";

psOutputs.SetProperty "IntObjectName","Sample Account";

psOutputs.SetProperty "MessageId", "";

psOutputs.SetProperty "MessageType", "Integration Object";

svcReadFile.InvokeMethod("ReadEAIMsg",psInputs, psOutputs);

svcSaveData.InvokeMethod("Upsert",psOutputs,psOutputs2);
```

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

```
Set Inp = TheApplication.NewPropertySet

Inp.SetProperty "FileName", "c:\test.xml"

Inp.SetProperty "DispatchService", "XML Converter"

Inp.SetProperty "DispatchMethod" , "XMLToPropSet"

Set svc = theApplication.GetService("EAI File Transport")

Set XMLOutputs = theApplication.NewPropertySet

svc.InvokeMethod "ReceiveDispatch", Inp, XMLOutputs

TheApplication.RaiseErrorText Cstr(XMLOutputs.GetChildCount)
```

Business Scenario for the Use of Business Services

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

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

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

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

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

```
<FirstName>Hector</FirstName>

<LastName>Alacon</LastName>
```

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

Code Sample Example for Creating a Property

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

```
x = TheApplication.InvokeMethod("WebForm", inputs, outputs);

var svc; // variable to contain the handle to the Service

var inputs; // variable to contain the XML input

var outputs; // variable to contain the output property set

svc = TheApplication().GetService("EAI XML Read from File");

   inputs = TheApplication().ReadEAIMsg("webform.xml");

   outputs = TheApplication().NewPropertySet();

svc.InvokeMethod("Read XML Hierarchy", inputs, outputs);
```

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

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

```
Function Service_PreInvokeMethod(MethodName, inputs, outputs)

   {
```

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

5 Web Services

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

About Web Services

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

Business services or workflow processes in Siebel Business Applications can be exposed as Web Services to be consumed by an application. Siebel Web Services framework has an ability to generate WSDL files to describe the Web Services hosted by the Siebel application. Also, the Siebel Web Services framework can call 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:
  - "About WS-Security UserName Token Profile Support" on page 93
  - http://schemas.xmlsoap.org/ws/2002/07/secext
- 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 the Siebel application, 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 the Siebel application 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.

About 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 may 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 applet following the instructions in "Invoking Siebel Web Services Using an External System" on page 76.
3. Create a new inbound service port record in the Service Ports list applet as indicated in "Invoking Siebel Web Services Using an External System" on page 76, and in the Binding column select SOAP_RPC_LITERAL from the drop-down list.

About 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 75. 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, it is a one-way operation.
- **Outbound.** If the service proxy method has no output arguments, 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 > 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 in order to invoke this Web Service.

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, make sure that ASI is enabled for external use in Siebel Tools.

To create an Inbound Web Service record

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

      NOTE: This step is required for generating various XML documents.

   b. Enter the name of the inbound Web Service in the Name column.
   c. Select Active in the Status field to enable external applications to call 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 applet:
   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, add a new type following Step c on page 77 through Step f on page 77; otherwise, move to Step g on page 77.

   c. Click New and select the implementation type (Business Service or Workflow).
   d. Select the implementation name (the business service or business process that implements the port type).
   e. Enter a name for the new type in the Name field and click Save.
   f. Click Pick in the Inbound Web Services Pick Applet to complete the process of adding a new Type.
   g. Select the protocol or transport 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 called, for example:

        `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: 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 applet, create a new operation record for the new service port you created in Step 3 on page 75 and want to publish:

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

   a. Enter the name of the Web Service operation.

   b. Select the name of the business service method in the Method Display Name column.

      NOTE: The Method Display Name column defaults to RunProcess if you have chosen Workflow Process in Step 3 on page 77 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 74.

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 invoke 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 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 VisualStudio.Net, use the wsdl.exe utility, for example, wsdl.exe /l:CS mywsdlfile.wsdl.

- In Apache's AXIS, use the wsdl2java utility, for example, java org.apache.axis.wsdl.WSDL2Java mywsdlfile.wsdl.
- In IBM's WSADIE, depending on the version, add the WSDL file to the Services perspective and run the Create Service Proxy wizard.

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

About Defining Web Services 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 containing XML data, converts the XML data to business service method arguments, and invokes the appropriate method for the appropriate service (business service or process). After the called method finishes its execution, the Web Service Inbound Dispatcher converts the output arguments to XML data, and returns the XML embedded in the SOAP envelope. During this process, any errors are returned as SOAP fault messages.

SOAP Fault Message Example

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

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

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

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

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 invoke 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 may involve defining VBCs based on the Web Service.

Creating an Outbound Web Service Based on a WSDL File

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

Data and methods for an outbound Web Service can be defined by either:

- A WDSL file for the external Web Service
- An outbound ASI

To create an outbound Web Service based on a WSDL file

1. Start Siebel Tools to create the proxy business service.
2. Create a new project and lock the project, or lock an existing project.
3. Choose File > New Object... to display the New Object Wizards.
4. Select the EAI tab and 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. Click Next to view and verify a summary of your import information.

   f. 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 CSSWSOutboundDisptacher 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, are created 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 (an XML file) containing the run-time Web Service administration data that should 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 81.

Outbound Web Services Administration

The WSDL Import Wizard exports the data to a file that you must import to the run-time database (the Web Services address) using the Outbound Web Services 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 need 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 bring up the EAI Web Service Import dialog box.
4. Specify the export file created by the Web Services Import Wizard.
5. Click Import to import the Web Service definition into the database.

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

To manually create a new outbound Web Service

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

   NOTE: When importing an external Web Service, you do not need to specify the proxy business service, integration objects, or the run-time parameters.
3. In the Service Ports list applet, create a new outbound service ports record:
   a. Enter the name of the Web Service port in the Name column.
   b. Select a transport name for the protocol or queuing system for the Transport.
   c. Enter the address appropriate for the transport chosen.
      - Enter the URL or queue that will publish the Web Service. The URL format to publish using HTTP is:

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

      Where:

      webserver = The machine name of the Siebel Web Server.

      lang = The default language of the Object Manager to handle the request.

      - The format to publish using JMS transport is:

        `jms://queue name@connection factory`

      Where:

      queue name = The JNDI name of the queue.

      connection factory = The JNDI name of the JMS connection factory.

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

      - For the Local Web Service transport, enter the name of the inbound port.
      - The format to publish over EAI MQSeries or EAI MSMQ transports is:

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

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

Where:

queue name = The name of the queue that is specified by either the EAI MQ Series, or the EAI MSMQ transports at the time of their design.

queue manager name = The name of the EAI MQSeries Transport queue manager.

queue machine name = The name of the machine 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 or the Local Business Service transports, enter the name of a Business Process or Business Service that should be called.

d. 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 Local Workflow or Local Business Service transport to avoid the overhead of processing XML.

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

4. In the Operations list applet, create a new operation record for the new service port you created in Step 3 on page 82:

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 92.

5. Generate the WSDL file. For information, see "Generating a WSDL File" on page 78.

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. Start Siebel Tools.
2. Select the outbound Web Service proxy business service you want to use to call your outbound Web Service.
3. Add the following user properties for this business service and set their values based on the outbound service port of your Web Service:

- siebel_port_name
- siebel_web_service_name
- siebel_web_service_namespace

Integration Objects as Input Arguments to Outbound Web Services

The property set that is used as an input argument to the outbound Web Service should have the same name as the input argument's 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 name from SiebelMessage to the actual outbound Web Service argument name by using a Siebel eScript service before calling the outbound Web Service.

Web Services Support for Transport Headers

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

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

- Name: siebel_transport_header:headerName
- Value: Header value

The following are examples of input arguments:

- Name: siebel_transport_header:UserDefinedHeader
- Value: myData
- Name: siebel_transport_header:Authorization
- Value: 0135DFDJKLJ

About Local Business Service

In many instances, Web Services use specialized SOAP headers for common tasks such as authentication, authorization, and logging. In order to support this common Web Service extensibility mechanism, a Local Business Service, as a transport option for outbound Web Services, is supported in the Siebel application. 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 24.

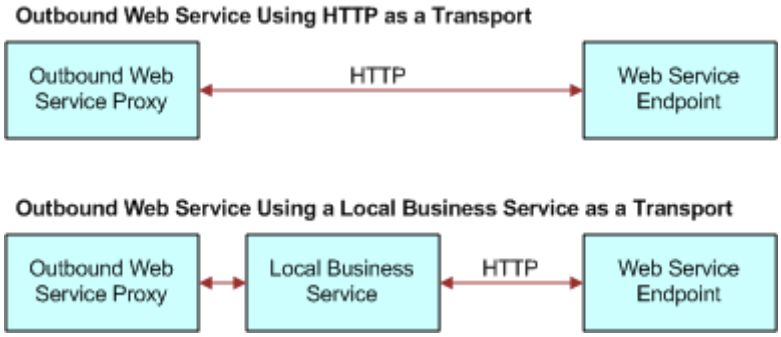


Figure 24. Local Business Service Used as a Transport

If the Web Service to be invoked is within the sample application, then no need exists to invoke 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, 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, the card information is passed through 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 may be added to address infrastructure requirements by direct modification of the input property, set by using Siebel eScript or Siebel VB.

About XML Schema Support for the <xsd:any> Tag

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

About Mapping the <xsd:any> Tag in the WSDL Import Wizard

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

The <xsd:any> tag can contain an attribute called *namespace*. If the value for that attribute is known, then one or more integration components or even an integration object can be created. If 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 an attribute called *namespace*. If the namespace value is known (the conditions for being known were noted in this section), then all the global attributes for that particular schema will be added in place of this tag. Therefore, one or more integration component fields can potentially be created.

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

About Mapping the <xsd:any> Tag in the XML Schema Wizard

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

The <xsd:any> tag can contain an attribute called *namespace*. If the value for that attribute is known, then one or more integration components or even an integration object can be created. If the value for that attribute is not known, 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 an attribute called *namespace*. If the namespace value is known (the condition for being known was noted in this section), then all the global attributes for that particular schema will be added in place of this tag. Therefore, one or more integration component fields can potentially be created.

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

Examples of Invoking Web Services

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

Invoking an External Web Service Using Workflow or Scripting

As illustrated on Figure 25 on page 88, the following steps are executed to invoke an external Web Service:

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

7 The SOAP response is converted to a property set that can be processed by the caller—for example, Calling Function.

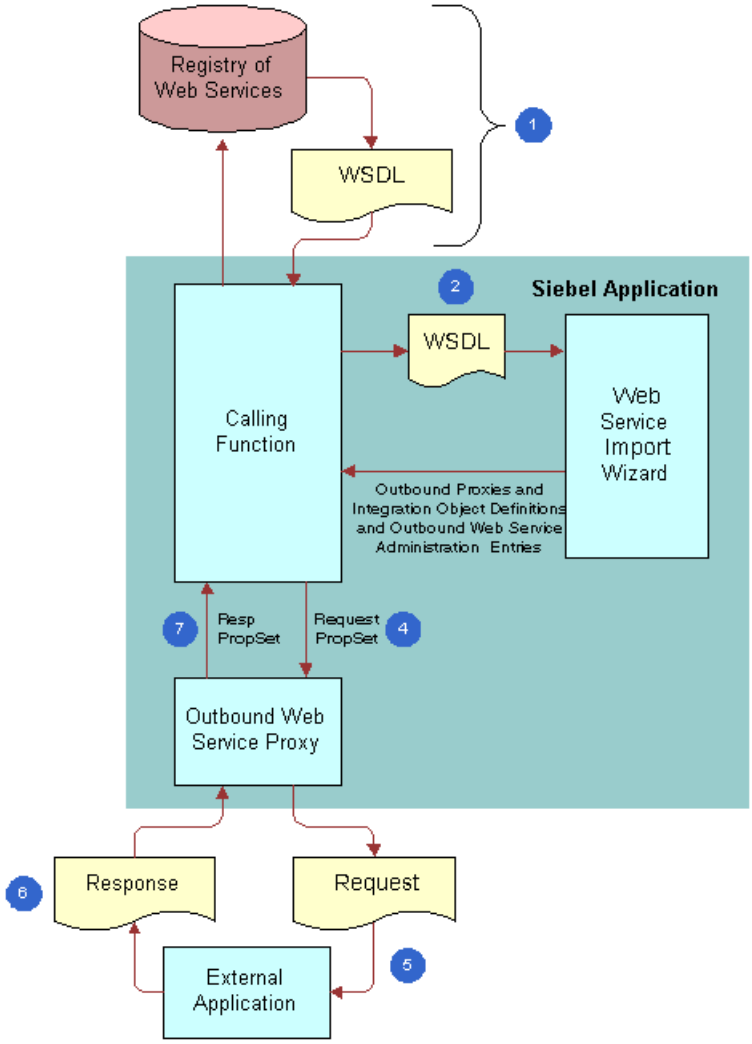


Figure 25. Invoking an External Web Service

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

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

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

```
if (MethodName == "invoke") {
    var svc = TheApplication().GetService("CustomerDBClientSimpleSoap");
    var wsInput = TheApplication().NewPropertySet();
    var wsOutput = TheApplication().NewPropertySet();
    var getCustInput =  TheApplication().NewPropertySet();
    var listOfGetCustomerName = TheApplication().NewPropertySet();
    var getCustomerName = TheApplication().NewPropertySet();

    try {
        // obtain the customer ID to query on. This value will be provided in the input property set
        var custId =  Inputs.GetProperty("custId");

        // set property to query for a customer ID with a value of '1'
        getCustomerName.SetType("getCustomerName");
        getCustomerName.SetProperty("custid", custId);

        // set Type for listOfGetCustomerName
        listOfGetCustomerName.SetType("ListOfgetCustomerName");

        // set Type for getCustInput
        getCustInput.SetType("getCustomerNameSoapIn:parameters");

        // assemble input property set for the service.
        listOfGetCustomerName.AddChild(getCustomerName);
        getCustInput.AddChild(listOfGetCustomerName);
        wsInput.AddChild(getCustInput);

        // invoke the getCustomerName operation
        svc.InvokeMethod("getCustomerName", wsInput, wsOutput);

        // parse the output to obtain the customer full name check the type element on each PropertySet
        (parent/child) to make sure we are at the element to obtain the customer name
        if (wsOutput.GetChildCount() > 0) {
           var getCustOutput = wsOutput.GetChild(0);
           if (getCustOutput.GetType() == "getCustomerNameSoapOut:parameters") {
              if (getCustOutput.GetChildCount() > 0) {
                 var outputListOfNames = getCustOutput.GetChild(0);
                 if (outputListOfNames.GetType() == "ListOfgetCustomerNameResponse") {
                    if (outputListOfNames.GetChildCount() > 0) {
                       var outputCustName = outputListOfNames.GetChild(0);
                       if (outputCustName.GetType() == "getCustomerNameResponse") {
                          var custName = outputCustName.GetProperty("getCustomerNameResult");
                          Outputs.SetProperty("customerName", custName);
                        }
                     }
                  }
              }
           }
        }

        return (CancelOperation);

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

About Invoking a Siebel Web Service from an External Application

As illustrated in Figure 26 on page 90, 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 in order to create a proxy that can be used to call 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 calling external application.

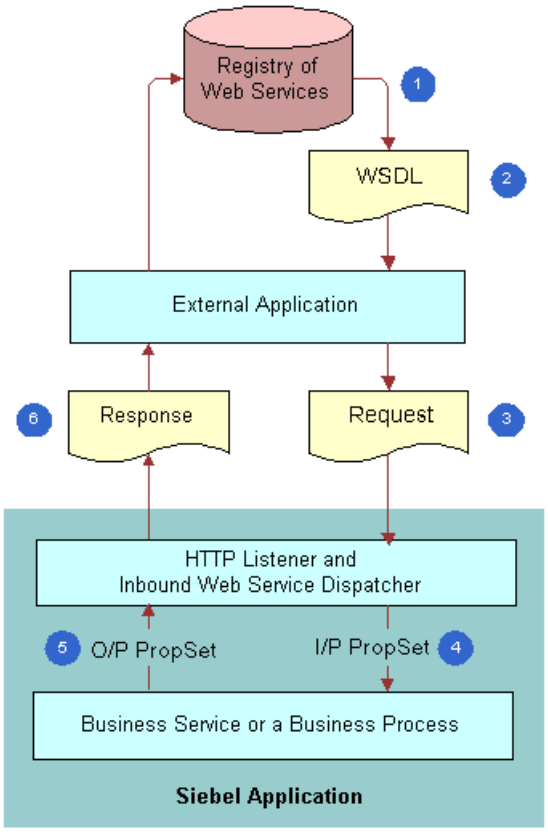


Figure 26. Invoking a Siebel Web Service

The following is an example of invoking Siebel published Web Service using .NET.

```
// removed using declaration

namespace sieOppClnt
```

```
{
 public class sieOppClnt : System.Web.Services.WebService
 {
    public siebOptyClnt()
    {
       InitializeComponent();
    }

    // WEB SERVICE CLIENT EXAMPLE

    // The optyQBE returns a list of opty based upon the required input params. Since
    the input to the Siebelopty.QueryByExample method uses an Input/Output param,
    ListOfInterOptyIntfaceTopElmt will be passed by ref to Siebel. To add the Siebel
    Opportunity Web Service definition to the project, I chose to run the wsdl.exe
    utility to generate the necessary helper C# class for the service definition.
    [WebMethod]
    public ListOfInterOptyIntfaceTopElmt optyQBE(string acctName, string acctLoc,
    string salesStage)
    {
       Siebelopty svc = new Siebelopty();
       ListOfInterOptyIntfaceTopElmt siebelMessage = new
       ListOfInterOptyIntfaceTopElmt();
       ListOfInteroptyInterface optyList = new ListOfInteroptyInterface();
       opty[] opty = new opty[1];
       opty[0] = new opty();
       opty[0].Account = acctName;
       opty[0].AccountLocation = acctLoc;
       opty[0].SalesStage = salesStage;

       //assemble input to be provided to the Siebel Web Service. For the sake of
       simplicity the client will query on the Account Name, Location, and Sales
       Stage. Ideally additional checking to make sure that correct data is entered.
       optyList.opty = opty;
       siebelMessage.ListOfInteroptyInterface = optyList;

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

       // return the raw XML of the result set returned by Siebel. Additional
       processing could be done to parse the response.
       return siebelMessage;
    }
 }
}
```

About Web Services Security Support

Siebel Systems endorses the industry standard known as the Web Services (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 *Security Guide for Siebel Business Applications*.

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. Create a named subsystem with the correct parameters. Do the following if creating this from the command line (command line example):

   a. Start the Siebel Server Manager.

      For information on accessing the Siebel Server Manager, see the *Siebel System Administration Guide*.

   b. Run the following command to create a subsystem named *SecureWebService* that is ready to be consumed by a WS-Security client:

      ```
      create named subsystem SecureWebService for subsystem
      EAITransportDataHandlingSubsys with DispatchService="Web Services Inbound
      Dispatcher",DispatchMethod="Dispatch",Impersonate="true"
      ```

3. When the client makes an actual call to the Web service, make sure that SWEExtSource is pointing to the correct virtual directory and named subsystem:

   ```
   http://myserver/eai_anon_enu/
   start.swe&SWEExtCmd=Execute&SWEExtSource=SecureWebService&UserName=user&Password=
   pass
   ```

About WS-Security UserName Token Profile Support

Siebel Business Applications support the WS-Security's 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.

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

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

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

The following is an example of passing the user name and password by way of a URL:

```
http://webserver/eai_enu/start.swe?SWEExtSource=WebService&SWEExtCmd=Execute&
Username=SADMIN&Password=SADMIN
```

With UserName tokens, the URL does not reveal the user credentials:

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

NOTE: Using WS-Security is optional. If security is of the utmost importance, and if it is critical that the password not be provided in clear text, use HTTPS.

About 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 need to be globally unique. The applet shown in Figure 27 can be defined as requiring no authentication, or requiring a UserName Token with username and password provided in clear text.

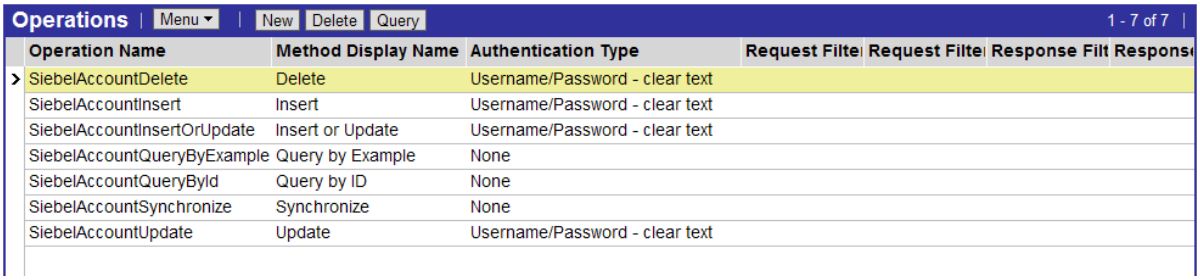
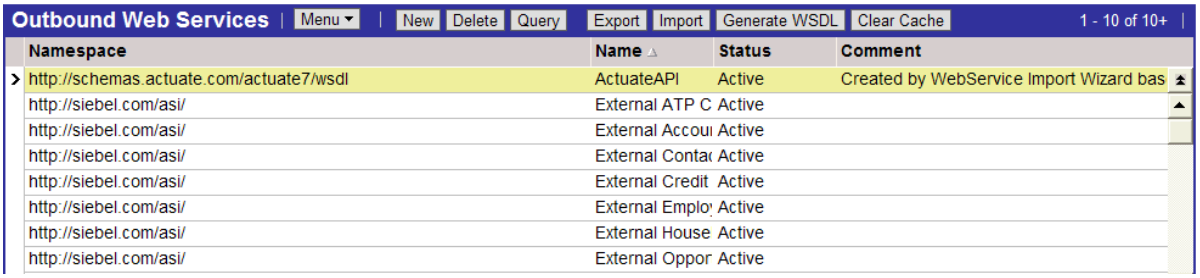


Figure 27. Inbound Web Services View and the UserName Token

NOTE: No authentication type implies that the user credentials are in the URL.

About Using the UserName Token for Outbound Web Services

Each Web Service operation in the Outbound Web Services list applet may be tied to an authentication type by selecting from the Authentication Type picklist (see Figure 28) in the Operations picklist, in the following applet.



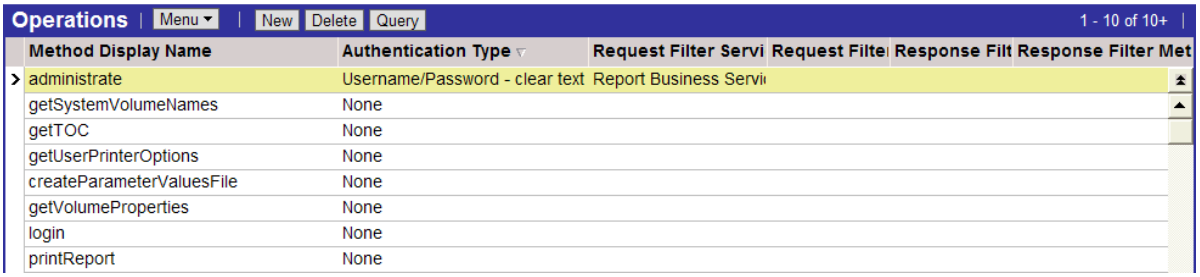


Figure 28. Outbound Web Services List Applet and the Operations PickList

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 invokes one of the following sessions:

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

For example, a custom Web application can send a request that includes a username and password, and invokes 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 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, a new session is created.

You can use the SOAP headers listed in Table 9 to invoke different types of sessions, and pass authentication credentials.

NOTE: The values entered are case insensitive.

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

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

Table 9. Siebel Session Management and Authentication SOAP Headers

SOAP Header Block	Description
SessionType	You use the SessionType SOAP header to define the type of session. Valid values are None, Stateless and Stateful: ■ **None.** A new session is opened for each request and then closed after a response is sent out. The SessionType none may or may not include UsernameToken and PasswordText SOAP headers. When UsernameToken and PasswordText SOAP headers are included, these credentials are used for authentication. If the UsernameToken and PasswordText SOAP headers are excluded from the SOAP header, anonymous login is assumed. The anonymous login requires additional configuration in the Siebel Web Engine (eapps.cfg) and Named Subsystem configuration (AllowAnonymous). For more information about configuring Anonymous login, see *Security Guide for Siebel Business Applications*. ■ **Stateless.** A new session is opened for an initial request and the session remains open for subsequent requests. Relogin occurs automatically (transparent to the user) if the session is closed. UsernameToken and PasswordText must be included as SOAP headers in the initial request to open a stateless session. ■ **Stateful.** A new 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. 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.

Table 9. Siebel Session Management and Authentication SOAP Headers

SOAP Header Block	Description
PasswordText	You use the PasswordText SOAP header to send the password used by the login ID to the Siebel server.
SessionToken	Session tokens are used with stateless requests. They are sent and received using the SessionToken SOAP header. After receiving an initial request with valid authentication credentials and a session type set to Stateless, the Siebel Server generates a session token and includes it in the SOAP header of the outbound response. The session token is encrypted and consists of a session ID and user credentials. The custom Web application uses the session token for subsequent requests. The Session Manager on the 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.

For examples of using SOAP headers for session management and authentication, see "Examples of Using SOAP Headers for Authentication and Session Management" on page 99.

CAUTION: Siebel Session Management and Authentication SOAP headers are supported on the following J2EE Application environments: Axis, BEA WebLogic, and IBM WebSphere.

NOTE: The Siebel Session Management and Authentication SOAP headers are different from the SOAP headers used for WS-Security. For more information about WS-Security, see "About WS-Security UserName Token Profile Support" on page 93.

Combinations of Session Types and Authentication Types

Table 10 summarizes the combinations of authentication types and session types.

Table 10. Summary of Authentication Types and Session Types

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

Table 10. Summary of Authentication Types and Session Types

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

For examples that illustrate some of these combinations, see "Examples of Using SOAP Headers for Authentication and Session Management" on page 99.

About Enabling the Session Management on SWSE

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

```
http://mywebserver/EAIObjMgr_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. For more information about WS-Security, see "About WS-Security UserName Token Profile Support" on page 93.

Session Time Out and Max Age Parameters

You control the session time out length and maximum age by setting the parameters listed in Table 11. These parameters are set in 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 11. Session Token TimeOut and MaxAge Parameters

Parameter Name	Parameter Value	Description
SesionTokenTimeout	Number in minutes	Number of minutes a session can remain inactive before the user is logged out and the session is closed.
SessionTokenMaxAge	Number in minutes	The total number of minutes a session can remain open before the user is logged out and the session is closed.

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 97.

Anonymous Request No Session

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

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

Siebel Authorization No Session

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

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

```
</soap:Header>
<soap:Body>
    <!-- data goes here -->
</soap:Body>
</soap:Envelope>
```

Siebel Authorization Stateless Session

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

Initial Request

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

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

Response

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

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

Subsequent Request Using Session Token

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

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header>
    <SessionType xmlns="http://siebel.com/webservices">Stateless</SessionType>
```

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

Simple Query Starting With <soap:body>

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

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

About Web Services and Web Single Sign-On Authentication

Siebel Web Services support Web single sign-on 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 29 illustrates a Web single sign-on deployment scenario using Siebel Web services. For more information about Web single sign-on, see *Security Guide for Siebel Business Applications*.

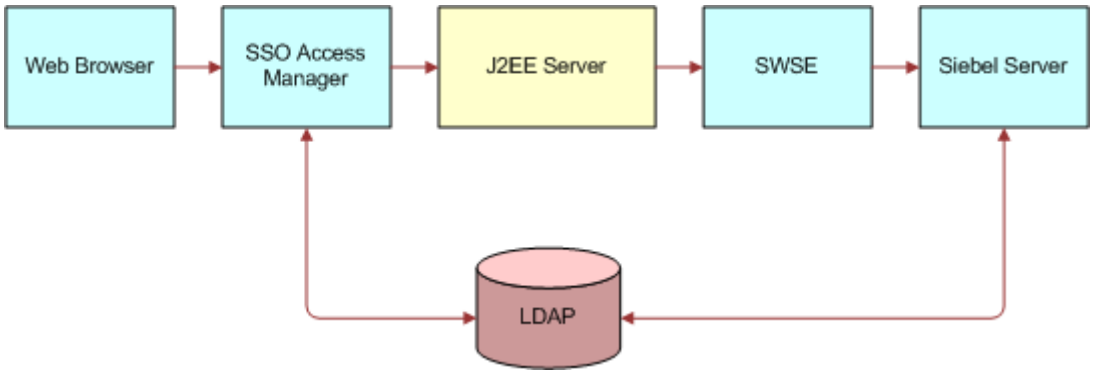


Figure 29. Web Single Sign-On Scenario

Each component in the SSO Scenario shown in Figure 29 is described below:

- **SSO Access Manager.** SSO Access Manager, configured in front of the J2EE server, challenges user login, authenticates user credentials with LDAP, and sets a security token in the browser (http header), which gets forwarded to the J2EE server.
- **J2EE 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 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.

About 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 may 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, 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, the custom header is not read and a SOAP:MustUnderstand fault is generated.
- If 'mustUnderstand' equals 0, no processing of the custom header is attempted.

You want to keep the SOAP body and header processing isolated. The inbound dispatcher and outbound proxy know how to process the SOAP body, but do not know how to 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: Headers that are consumed by the filter service have to be removed from the SOAP message.

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 invoke 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 invoke, and also the corresponding input and output parameters.

```
function Service_PreInvokeMethod (MethodName, Inputs, Outputs)
{
  if(MethodName == "StripHeader")
  {
       if(Inputs.GetChildCount() > 0)
       {
          Outputs.InsertChildAt(Inputs.GetChild(0), 0);
          var soapEnv = Outputs.GetChild(0);
          if(soapEnv.GetChildCount() == 2) // headers and body
          {
             var callBackHeader = soapHeader.GetChild(0);
             if(callBackHeader.GetChildCount() == 2)
```

```
            {
               var headerContext = TheApplication().NewPropertySet();
               headerContext.SetType("HeaderContext");
               // get the header child property set
               var callBackLocnHeader = callBackHeader.GetChild(0);
               var correlationIdHeader = callBackHeader.GetChild(1);
               headerContext.AddChild(callBackLocnHeader);
               headerContext.AddChild(correlationIdHeader);
               soapHeader.RemoveChild(0);
               Outputs.AddChild(headerContext);
            }
         }
      }
   }
   else if(MethodName == "AddHeader")
   {
       if(Inputs.GetChildCount() > 0)
       {
           Outputs.InsertChildAt(Inputs.GetChild(0), 0);
           var soapEnv = Outputs.GetChild(0);
                 var soapHeader = TheApplication().NewPropertySet();
                 soapHeader.SetType("soapEnv:Header");
                 soapHeader.SetProperty("xmlns:soapEnv", "http://schemas.xmlsoap.org/soap/envelope/");
                 var correlationIdHeader = TheApplication().NewPropertySet();
                 correlationIdHeader.SetType("CorrelationId");
       if(Inputs.GetChildCount() == 2)
       {
           // get the correlation id from soap header context
           var soapHeaderCntxt = Inputs.GetChild(1);
           var corIdHeader = soapHeaderCntxt.GetChild(0);
           correlationIdHeader.SetValue(corIdHeader.GetValue());
           }
           else
           {
              // set default correlation id header
              correlationIdHeader.SetValue("30");
           }
           soapHeader.AddChild(correlationIdHeader);
           soapEnv.InsertChildAt(soapHeader, 0);|
       }
   }
   else if(MethodName == "AddPSHeader")
   {
       if(Inputs.GetChildCount() > 0)
       {
           Outputs.InsertChildAt(Inputs.GetChild(0), 0);
           var soapEnv = Outputs.GetChild(0);
                 var soapHeader = TheApplication().NewPropertySet();
                 soapHeader.SetType("PropertySetHeader");
                 soapHeader.SetProperty("xmlns:PropertySet", "http://www.siebel.com/propertyset");
                           var correlationIdHeader = TheApplication().NewPropertySet();
                           correlationIdHeader.SetType("CorrelationId");
                 if(Inputs.GetChildCount() == 2)
                 {
                   // get the correlation id from soap header context
                   var corIdHeader = soapHeaderCntxt.GetChild(0);
                   correlationIdHeader.SetValue(corIdHeader.GetValue());
                 }
                 else
                 {
                   // set default correlation id header
                   correlationIdHeader.SetValue("30");
                 }
                 soapHeader.AddChild(correlationIdHeader);
                 soapEnv.InsertChildAt(soapHeader, 0);
       }
   }
           return (CancelOperation);
}
```

About 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 invoked the *HeaderContext* property set is passed to the business service or workflow that is invoked. 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. In order to use it in the business service or workflow execution call, this information has to be contained in the *HeaderContext*. Internally, the Siebel Web Services infrastructure passes *HeaderContext* to the eventual business service or workflow that is invoked.

About Web Services Cache Refresh

Both Siebel Inbound and Outbound Web Services are typically cached into memory on the Siebel Server. At times, administrators need to 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 Siebel services.

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 requires user interaction. The administrator decides when the Web Service configuration must be refreshed. When used, Web Service configuration changes can be made without restarting the Siebel Server or the Server Component that uses the configuration.

The Clear Cache feature is a button on the Administration - Web Services screen. This feature is available for inbound and outbound Web Services.

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.

4. Choose the relevant application object manager.

5 In the bottom applet, click the Parameters tab.

This list applet contains the parameters for the Component selected in the middle applet.

6 Set the Log Level to 4 for any or all of the following Event Types.

Event Type	Alias	Description	Comment
Web Service Performance	WebSvcPerf	Web Service Performance Event Type	Used for performance logging.
Web Service Outbound Argument Tracing	WebSvcOutboundArgTrc	Web Service Outbound Run-time Argument Tracing	Used for logging arguments to the outbound dispatcher.
Web Service Outbound	WebSvcOutbound	Web Service Outbound Run-time Event Type	Used for runtime logging of outbound Web Services.
Web Service Loading	WebSvcLoad	Web Service Configuration Loading Event Type	Used for logging of the loading of Web Services.
Web Service Inbound Argument Tracing	WebSvcInboundArgTrc	Web Service Inbound Run-time Argument Tracing	Used for logging arguments to the inbound dispatcher.
Web Service Inbound	WebSvcInbound	Web Service Inbound Run-time Event Type	Used for logging at Web Service inbound runtime. Information is logged to the inbound dispatcher.
Web Service Design	WebSvcDesign	Web Service Design-time Event Type	Used for logging at Web Service design time. For example, at the time of WSDL import and generation.

7 Navigate to the Components view.

8 Select the EAI Object Manager component, and select the Component Parameters tab.

9 Set the Enable Business Service Argument Tracing parameter to True.

10 Restart or reconfigure the server component. For information, see the *Siebel System Administration Guide*.

About the Cardinality of the Root Integration Components

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

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

NOTE: When modifying run-time parameters, restart the server component. For information, see the *Siebel System Administration Guide*.

6 EAI Siebel Adapter

The 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, and the different methods and arguments you can use with the EAI Siebel Adapter to manipulate the data in the Siebel Database.

The following topics are included:

About the EAI Siebel Adapter

The EAI Siebel Adapter is a general purpose integration business service that allows you to:

- Read Siebel business objects from the Siebel Database into integration objects.
- Write an integration object 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 123.

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

Node Types and the EAI Siebel Adapter

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.

For more information on node types, see “About the EAI UI Data Adapter Business Service” on page 149.

EAI Siebel Adapter Methods

The EAI Siebel Adapter supports the following methods:

About the Examples in the EAI Siebel Adapter 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 an XML format.
- The XML document or integration object instance may also be referred to as a Siebel Message.
- Fields that contain null values are not included in the XML examples.

  However, these fields may 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.

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" on page 138.

When using the EAI Siebel Adapter, to query all the business component records, you do not need to specify any value in the Object Id process property of the workflow process. In this case not specifying an 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 criteria, then all the fields present in the QBE will be used in the query. To retrieve a unique record, include the fields that make up the user key for the underlying integration object component instance to ensure you retrieve a unique record. You can use an asterisk (*) as a wildcard for each one of the fields.

For example, the following is your QBE:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2IOY" IntObjectName = "EAI Account" MessageType =
"Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfAccount>
    <Account>
       <CSN>*</CSN>
       <HomePage>*</HomePage>
       <Location>H*</Location>
       <Name>A*</Name>
       <Type>*</Type>
    </Account>
 </ListOfAccount>
</SiebelMessage>
```

You would receive all of the Accounts with names that start with A* and have locations that start with H*. The CSN, HomePage, and Type fields cannot be blank because they are used in the query.

The EAI Siebel Adapter converts the QBE into a user Search Expression of the following:

```
[CSN] ~ LIKE "*" AND [Home Page] ~ LIKE "*" AND [Location] ~ LIKE "H*" AND [Name] ~
LIKE "A*" AND [Type] ~ LIKE "*"
```

You can run this example and review the output XML generated.

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, use the QueryPage Method. You can combine the Object Id and Search Specification together to query for parent and child data.

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 23 on page 135. You can also use method arguments such as OutputIntObjectName, SearchSpec, SortSpec, ViewMode, and StartRowNum to dictate which records to be returned.

Even though the QueryPage returns a limited number of records, it keeps the data in the cache, which you can then retrieve by calling the EAI Siebel Adapter with a new value for the StartRowNum method argument. Please note that this is only possible if the method arguments OutputIntObjectName, PageSize, SearchSpec, SortSpec, and ViewMode are not changed and the NewQuery method argument is set to False.

NOTE: This is for both the Query( ) and QueryPage( ) method: The EAI Siebel Adapter returns the outputs of both methods as one Siebel Message. This integration object instance is stored in the process memory. If your query returns a large number of messages, this will result in your Siebel component's memory consumption being high.

The following is an example of using the QueryPage( ) method in a business service.

```
var EAIService = TheApplication().GetService("EAI Siebel Adapter");
var writeSvc = TheApplication().GetService("EAI XML Write to File");
var EAIin = TheApplication().NewPropertySet();
var ResultSet= TheApplication().NewPropertySet();
var moreRecords = true;
var countOfObjects  = 0;
var i = 1;

// set up input arguments, get 10 at a time
EAIin.SetProperty("OutputIntObjectName", "EAI Account");
EAIin.SetProperty("PageSize", "10");
EAIin.SetProperty("SearchSpec", "[Account.Name] LIKE '3*'");
EAIin.SetProperty("StartRowNum", i);
EAIin.SetProperty("NewQuery", "true");

// retrieve the business component data
EAIService.InvokeMethod("QueryPage", EAIin, ResultSet);

// loop through cached data
while ( (ResultSet.GetChildCount() > 0) && (moreRecords)) {
   countOfObjects = countOfObjects + ResultSet.GetProperty("NumOutputObjects");
```

```
    // write out first chunk of dta retrieved
    ResultSet.SetProperty("FileName", "d:\\temp\\EAIaccount$$.xml”);
    writeSvc.InvokeMethod("WriteEAIMsg", ResultSet, Outputs);

    // reuse the existing input property set, except don't reissue query
    EAIin.SetProperty("NewQuery", "false");
    i= i+10;  // get next 10 records
    EAIin.SetProperty("StartRowNum", i);

    ResultSet.Reset(); // clear previous result set
    EAIService.InvokeMethod("QueryPage", EAIin, ResultSet);

    if (ResultSet.GetProperty("LastPage") == "true")
       moreRecords = false;

}
```

Synchronize Method

You can use the Synchronize method to make the values in a business object instance match those of an integration object instance. This operation can result in updates, insertions, or deletions in the business components. The following rules apply to the results of this method:

- If a child component is not present in the integration object instance, 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.

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

Example of Synchronize Method on Deleted Unmatched Children

This first example demonstrates deleting unmatched children when using the Synchronize method. This example uses data present in the sample database.

```
<?xml version = "1.0" encoding = "UTF-8"?><?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType =
"Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfAccount>
    <Account>
       <AccountStatus>Active</AccountStatus>
       <CSN>1-3JGO7</CSN>
       <CurrencyCode>USD</CurrencyCode>
       <LanguageCode>ENU</LanguageCode>
```

```
<Location>Test</Location>
<Name>ABC Corp</Name>
<ListOfAccount_BusinessAddress>
      <Account_BusinessAddress IsPrimaryMVG = "Y">
         <AddressActiveStatus>Y</AddressActiveStatus>
         <BillAddressFlag>Y</BillAddressFlag>
         <City>ATown</City>
         <Country>USA</Country>
         <MainAddressFlag>Y</MainAddressFlag>
         <ShipAddressFlag>Y</ShipAddressFlag>
         <StreetAddress>123 Main St</StreetAddress>
      </Account_BusinessAddress>
      <Account_BusinessAddress IsPrimaryMVG = "N">
         <AddressActiveStatus>Y</AddressActiveStatus>
         <BillAddressFlag>Y</BillAddressFlag>
         <City>BTown</City>
         <Country>USA</Country>
         <MainAddressFlag>Y</MainAddressFlag>
         <ShipAddressFlag>Y</ShipAddressFlag>
         <StreetAddress>456 Second Street</StreetAddress>
      </Account_BusinessAddress>
</ListOfAccount_BusinessAddress>
<ListOfContact>
      <Contact>
         <ActiveStatus>Y</ActiveStatus>
         <FirstName>User1</FirstName>
         <LastName>User1</LastName>
         <Organization>Default Organization</Organization>
         <ListOfContact_Organization>
            <Contact_Organization IsPrimaryMVG = "Y">
               <Organization>Default Organization</Organization>
               <OrganizationIntegrationId/>
            </Contact_Organization>
         </ListOfContact_Organization>
         <ListOfContact_AlternatePhone/>
      </Contact>
      <Contact>
         <ActiveStatus>Y</ActiveStatus>
         <FirstName>User2</FirstName>
         <LastName>User2</LastName>
         <Organization>Default Organization</Organization>
         <ListOfContact_Organization>
            <Contact_Organization IsPrimaryMVG = "Y">
               <Organization>Default Organization</Organization>
               <OrganizationIntegrationId/>
            </Contact_Organization>
         </ListOfContact_Organization>
         <ListOfContact_AlternatePhone/>
      </Contact>
   </ListOfContact>
   <ListOfAccount_Organization>
      <Account_Organization IsPrimaryMVG = "Y">
         <Organization>Default Organization</Organization>
         <OrganizationId>0-R9NH</OrganizationId>
```

```
                <OrganizationIntegrationId/>
              </Account_Organization>
          </ListOfAccount_Organization>
      </Account>
   </ListOfAccount>
</SiebelMessage>
```

Then the following XML (integration object instance) is submitted with Synchronize:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType =
"Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfAccount>
    <Account>
       <AccountStatus>Active</AccountStatus>
       <CSN>1-3JGO7</CSN>
       <Competitor>Y</Competitor>
       <CurrencyCode>USD</CurrencyCode>
       <LanguageCode>CHS</LanguageCode>
       <Location>test</Location>
       <Name>ABC Corp</Name>
       <ListOfContact>
             <Contact>
                <ActiveStatus>N</ActiveStatus>
                <FirstName>User1</FirstName>
                <LastName>User1</LastName>
                <MiddleName></MiddleName>
                <Organization>Default Organization</Organization>
                </Contact>
             <Contact>
                <FirstName>User3</FirstName>
                <LastName>User3</LastName>
                <MiddleName></MiddleName>
                <Organization>Default Organization</Organization>
                </Contact>
       </ListOfContact>
    </Account>
 </ListOfAccount>
</SiebelMessage>
```

Following is the result you will receive. Because the contact information is included in the integration object instance, User2 in the database is deleted because it was an unmatched node. User1 is updated because it is a matched node. User3 is inserted because it is a new node. Since Business Address was not included in the integration object instance, it is left in the business object.

```
<?xml version = "1.0" encoding = "UTF-8"?><?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType =
"Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfAccount>
    <Account>
       <AccountStatus>Active</AccountStatus>
       <CSN>1-3JGO7</CSN>
       <CurrencyCode>USD</CurrencyCode>
```

```
            <LanguageCode>CHS</LanguageCode>
            <Location>Test</Location>
            <Name>ABC Corp</Name>
            <ListOfAccount_BusinessAddress>
                  <Account_BusinessAddress IsPrimaryMVG = "Y">
                     <AddressActiveStatus>Y</AddressActiveStatus>
                     <BillAddressFlag>Y</BillAddressFlag>
                     <City>ATown</City>
                     <Country>USA</Country>
                     <MainAddressFlag>Y</MainAddressFlag>
                     <ShipAddressFlag>Y</ShipAddressFlag>
                     <StreetAddress>123 Main St</StreetAddress>
                  </Account_BusinessAddress>
                  <Account_BusinessAddress IsPrimaryMVG = "N">
                     <AddressActiveStatus>Y</AddressActiveStatus>
                     <BillAddressFlag>Y</BillAddressFlag>
                     <City>BTown</City>
                     <Country>USA</Country>
                     <MainAddressFlag>Y</MainAddressFlag>
                     <ShipAddressFlag>Y</ShipAddressFlag>
                     <StreetAddress>456 Second Street</StreetAddress>
                  </Account_BusinessAddress>
               </ListOfAccount_BusinessAddress>
               <ListOfContact>
                  <Contact>
                     <ActiveStatus>N</ActiveStatus>
                     <FirstName>User1</FirstName>
                     <LastName>User1</LastName>
                     <Organization>Default Organization</Organization>
                     <ListOfContact_Organization>
                        <Contact_Organization IsPrimaryMVG = "Y">
                           <Organization>Default Organization</Organization>
                           <OrganizationIntegrationId/>
                        </Contact_Organization>
                     </ListOfContact_Organization>
                     <ListOfContact_AlternatePhone/>
                  </Contact>
                  <Contact>
                     <ActiveStatus>N</ActiveStatus>
                     <FirstName>User3</FirstName>
                     <LastName>User3</LastName>
                     <Organization>Default Organization</Organization>
                     <ListOfContact_Organization>
                        <Contact_Organization IsPrimaryMVG = "Y">
                           <Organization>Default Organization</Organization>
                           <OrganizationIntegrationId/>
                        </Contact_Organization>
                     </ListOfContact_Organization>
                     <ListOfContact_AlternatePhone/>
                  </Contact>
               </ListOfContact>
            </Account>
         </ListOfAccount>
      </SiebelMessage>
```

Table 12 is a high-level representation of the previous example.

Table 12. Representation of the Synchronize Method on Deleted Unmatched Children

Record In Database	Integration Object Instance	Record After Synchronize
Account: ABC Corp BusinessAdress: 123 Main St. BusinessAdress: 456 Second St. Contact: User1 Organization: Default Org. Contact: User2 Organization: Default Org. Organization: Default Org.	Account: ABC Corp Contact: User1 Contact: User3	Account: ABC Corp BusinessAdress: 123 Main St. BusinessAdress: 456 Second St. Contact: User1 Organization: Default Org. Contact: User3 Organization: Default Org. Organization: Default Org.

This second example demonstrates how all records with an empty container are deleted when using the Synchronize method.

If you start with this business component data:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType =
"Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfAccount>
    <Account>
       <AccountStatus>Active</AccountStatus>
       <CSN>1-3JGO7</CSN>
       <CurrencyCode>USD</CurrencyCode>
       <LanguageCode>ENU</LanguageCode>
       <Location>test</Location>
       <Name>ABC Corp</Name>
       <ListOfAccount_BusinessAddress>
             <Account_BusinessAddress IsPrimaryMVG = "Y">
                <AddressId>1-3JGOA</AddressId>
                <AddressActiveStatus>Y</AddressActiveStatus>
                <BillAddressFlag>Y</BillAddressFlag>
                <City>MyTown</City>
                <Country>Canada</Country>
                <MainAddressFlag>Y</MainAddressFlag>
                <ShipAddressFlag>Y</ShipAddressFlag>
                <StreetAddress>123 Main St</StreetAddress>
```

```
            </Account_BusinessAddress>
            <Account_BusinessAddress IsPrimaryMVG = "N">
               <AddressActiveStatus>Y</AddressActiveStatus>
               <BillAddressFlag>Y</BillAddressFlag>
               <AddressId>1-3JGOB</AddressId>
               <City>YourTown</City>
               <Country>Canada</Country>
               <MainAddressFlag>Y</MainAddressFlag>
               <ShipAddressFlag>Y</ShipAddressFlag>
               <StreetAddress>456 Second Street</StreetAddress>
            </Account_BusinessAddress>
         </ListOfAccount_BusinessAddress>
         <ListOfContact>
            <Contact>
               <ActiveStatus>Y</ActiveStatus>
               <FirstName>User1</FirstName>
               <LastName>User1</LastName>
              <MiddleName/>
               <Organization>Default Organization</Organization>
               <ListOfContact_Organization>
                  <Contact_Organization IsPrimaryMVG = "Y">
                     <Organization>Default Organization</Organization>
                     <OrganizationIntegrationId/>
                  </Contact_Organization>
               </ListOfContact_Organization>
               <ListOfContact_AlternatePhone/>
            </Contact>
            <Contact>
               <ActiveStatus>Y</ActiveStatus>
               <FirstName>User2</FirstName>
               <LastName>User2</LastName>
               <MiddleName/>
               <Organization>Default Organization</Organization>
               <ListOfContact_Organization>
                  <Contact_Organization IsPrimaryMVG = "Y">
                     <Organization>Default Organization</Organization>
                     <OrganizationIntegrationId/>
                  </Contact_Organization>
               </ListOfContact_Organization>
               <ListOfContact_AlternatePhone/>
            </Contact>
         </ListOfContact>
         <ListOfAccount_Organization>
            <Account_Organization IsPrimaryMVG = "Y">
               <Organization>Default Organization</Organization>
               <OrganizationId>0-R9NH</OrganizationId>
               <OrganizationIntegrationId/>
            </Account_Organization>
         </ListOfAccount_Organization>
      </Account>
   </ListOfAccount>
</SiebelMessage>
```

And the following integration object instance is passed in:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType =
"Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfAccount>
      <Account>
        <AccountStatus>Active</AccountStatus>
        <CSN>1-3JGO7</CSN>
        <Competitor>Y</Competitor>
        <CurrencyCode>USD</CurrencyCode>
        <LanguageCode>CHS</LanguageCode>
        <Location>test</Location>
        <Name>ABC Corp</Name>
     <ListOfContact/>
     </Account>
 </ListOfAccount>
</SiebelMessage>
```

After the sync operation, all the children contacts are deleted because none of the nodes match.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType =
"Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfAccount>
    <Account>
       <AccountStatus>Active</AccountStatus>
       <CSN>1-3JGO7</CSN>
       <CurrencyCode>USD</CurrencyCode>
       <LanguageCode>ENU</LanguageCode>
       <Location>test</Location>
       <Name>ABC Corp</Name>
       <ListOfAccount_BusinessAddress>
             <Account_BusinessAddress IsPrimaryMVG = "Y">
                <AddressId>1-3JGOA</AddressId>
                <AddressActiveStatus>Y</AddressActiveStatus>
                <BillAddressFlag>Y</BillAddressFlag>
                <City>MyTown</City>
                <Country>Canada</Country>
                <MainAddressFlag>Y</MainAddressFlag>
                <ShipAddressFlag>Y</ShipAddressFlag>
                <StreetAddress>123 Main St</StreetAddress>
             </Account_BusinessAddress>
             <Account_BusinessAddress IsPrimaryMVG = "N">
                <AddressActiveStatus>Y</AddressActiveStatus>
                <BillAddressFlag>Y</BillAddressFlag>
                <AddressId>1-3JGOB</AddressId>
                <City>YourTown</City>
                <Country>Canada</Country>
                <MainAddressFlag>Y</MainAddressFlag>
                <ShipAddressFlag>Y</ShipAddressFlag>
                <StreetAddress>456 Second Street</StreetAddress>
             </Account_BusinessAddress>
       </ListOfAccount_BusinessAddress>
```

```
          <ListOfAccount_Organization>
                <Account_Organization IsPrimaryMVG = "Y">
                   <Organization>Default Organization</Organization>
                   <OrganizationId>0-R9NH</OrganizationId>
                   <OrganizationIntegrationId/>
                </Account_Organization>
          </ListOfAccount_Organization>
       </Account>
    </ListOfAccount>
  </SiebelMessage>
```

Table 13 is a high-level representation of the operation.

This second example demonstrates how all records with an empty container are deleted when using the Synchronize method.

Table 13. Representation of Records with Empty Containers Being Deleted Using Synchronize Method

Record In Database	Integration Object Instance	Record After Synchronize
Account: ABC Corp BusinessAdress: 123 Main St. BusinessAdress: 456 Second St. Contact: User1 Organization: Default Org. Contact: User2 Organization: Default Org. Organization: Default Org.	Account: ABC Corp Contact:	Account: ABC Corp BusinessAdress: 123 Main St. BusinessAdress: 456 Second St. Organization: Default Org.

Insert Method

This method is also similar to the Synchronize method with the exception that the EAI Siebel Adapter generates an error if a matching root component is found; otherwise, it inserts the root component and synchronizes all the children. It is important to note that when you insert a record, there is a possibility that the business component would create default children for the record, which need to be removed by the Insert method. The Insert method synchronizes the children, which deletes all the default children. For example, if you insert an account associated with a specific organization, it will also be automatically associated with a default organization. As part of the Insert method, the EAI Siebel Adapter deletes the default association, and associates the new account with only the organization that was originally defined in the input integration object instance. The EAI Siebel Adapter achieves this by synchronizing the children.

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, 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>
```

In order for this example to work, you will need to have the contact, Kastrup-Larsen, in the database. (The record exists in the sample database). If you try the insert method against a server database where the contact does not exist, 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, you will receive the following error message:

```
IDS_ERR_EAI_SA_INSERT_MATCH_FOUND. Insert operation on integration component
'Service Request' failed because a matching record in business component 'Service
Request' with search specification '[SR Number] = "1-MYUNIQUEVALUE" was found.(SBL-
EAI-04383).
```

Upsert Method

The Upsert method is similar to the Synchronize method with one exception; the Upsert method does not delete any records.

The Upsert method will result in insert or update operations. If the record exists, it will be updated. If the record does not exist, 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, the new record will be inserted. Once again, upsert will not delete any children. If existing children are found, they are updated.

The possibility exists to update multiple corresponding top-level parent business component records using one XML file. The following XML file illustrates how.

```
<SiebelMessage MessageId="" MessageType="Integration Object"
IntObjectName="Transaction">
 <ListofTransaction>
    <Transaction>
       <Field1>xxxx</Field1>
       <Field2>yyyy</Field2>
       .....
    </Transaction>
    <Transaction>
       <Field1>aaaa</Field1>
```

```
        <Field2>bbbb</Field2>
        .....
     </Transaction>
     ..............
  </ListofTransaction>
</SiebelMessage>
```

Update Method

This method is similar to the Synchronize method, except that the EAI Siebel Adapter returns an error if no match is found for the root component; otherwise, it updates the matching record and synchronizes all the children. For example, if you send an order with one order item to the EAI Siebel Adapter, it will take the following actions:

1. Queries for the order, and if it finds a match, it updates the record.
2. Updates or inserts the new order item depending on 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 invoke 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 the following XML document, the Account *Test account* is deleted:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2IOY" IntObjectName = "EAI Account" MessageType =
"Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfAccount>
    <Account>
       <Name>Test account</Name>
       <Location>EMV</Location>
    </Account>
```

```
    </ListOfAccount>
  </SiebelMessage>
```

Any child contacts that once belonged to the account will still remain in the database, but will not be associated with this Account.

Execute Method

The Execute method can be specified on the EAI Siebel Adapter to perform combinations of various operations on components in an integration object instance. This method uses the following operations:

- query
- querypage (same as query when used as children operation)
- sync (the same method as Synchronize and is the default operation)
- upsert
- update
- updatesync
- insert
- insertsync
- delete
- skipnode
- skiptree
- none

NOTE: A none operation is equivalent to operation sync.

These operations perform the same tasks as the related methods. For example, the delete operation makes the EAI Siebel Adapter delete the business component record matched to the particular integration component instance. However, what will be done to the children depends on the combination of the parent operation and the child operation. For information, see Table 14 on page 125.

NOTE: The operation method names are case sensitive. If you misspell an operation method, 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 needs to be inserted, updated, deleted, synchronized, and so on. These operations can be specified as an attribute at the component level. They cannot be specified for any other element.

The following XML example demonstrates using the upsert and delete operation to delete a particular child record without updating the parent:

```
<SiebelMessage MessageId="" MessageType="Integration Object" IntObjectName="Sample
Account">
   <ListofSampleAccount>
      <Account operation="upsert">
         <Name>A. K. Parker Distribution</Name>
         <Location>HQ-Distribution</Location>
         <Organization>North American Organization</          Organization>
         <Division/>
         <CurrencyCode>USD</CurrencyCode>
         <Description>This is the key account in the AK Parker
         Family</Description>
         <HomePage>www.parker.com</HomePage>
         <LineofBusiness>Manufacturing</LineofBusiness>
         <ListOfContact>
         <Contact operation="delete">
            <FirstName>Stan</FirstName>
            <JobTitle>Senior Mgr of MIS</JobTitle>
            <LastName>Graner</LastName>
            <MiddleName>A</MiddleName>
            <PersonalContact>N</PersonalContact>
         <Account>A. K. Parker Distribution</Account>
         <AccountLocation>HQ-Distribution</AccountLocation>
         </Contact>
         </ListOfContact>
      </Account>
   </ListofSampleAccount>
</SiebelMessage>
```

About Execute Method Operations

Specify an attribute named operation, in lowercase, to the component's XML element. The legal values for this attribute are upsert, sync, delete, query, update, insert, updatesync, insertsync, skipnode, skiptree, and none. If the operation is not specified on the root component, 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 14 represents the results of the operation on the current node.

Table 14. Operation Outcomes for the Child Node

Operation	What happens to the current node	What happens to unmatched children of current node
upsert	Update or insert	Leave alone
sync	Update or insert	Delete
update	Update	Delete

Table 14. Operation Outcomes for the Child Node

Operation	What happens to the current node	What happens to unmatched children of current node
updatesync	Update	Delete
insert	Insert	Leave alone
insertsync	Insert	Delete
skipnode	Skip this node	Leave alone
skiptree	Skip the tree	Not applicable

Example of a Parent Node Using a Sync Operation

This example demonstrates the effects of records after a sync operation is performed. Table 15 is a high level representation of a parent node using the sync operation of the Execute method.

Table 15. Representation of a Parent Node Using the Sync Operation

Record In Database	Integration Object Instance	Record After Execute Operation
Account1	Account1 operation=sync	Account1
Contact0	Contact1	Contact1
Contact1	Contact2	Contact2

Record in Database

The code represents GENCOMM0 and GENCOMM1 being retrieved as the contacts for this example.

```
<?xml version = "1.0" encoding = "UTF-8"?><?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType =
"Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfAccount>
    <Account>
       <AccountStatus>Active</AccountStatus>
       <CurrencyCode>USD</CurrencyCode>
       LanguageCode>ENU</LanguageCode>
       <Location>San Francisco, CA</Location>
       <Name>GenComm</Name>
       <ListOfContact>
         <Contact>
           <FirstName>GENCOMM0</FirstName>
           <LastName>GENCOMM0</LastName>
           <MiddleName/>
           <Organization>Default Organization</Organization>
         </Contact>
         <Contact>
           <FirstName>GENCOMM1</FirstName>
           <LastName>GENCOMM1</LastName>
```

```
            <MiddleName/>
            <Organization>Default Organization</Organization>
          </Contact>
        </ListOfContact>
      /Account>
  </ListOfAccount>
</SiebelMessage>
```

Integration Object Instance

The following code represents the sync operation acting on the contacts from the database.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType =
"Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfAccount>
    <Account operation="sync">
       <AccountStatus>Inactive</AccountStatus>
       <CurrencyCode>USD</CurrencyCode>
       <LanguageCode>ENU</LanguageCode>
       <Location>San Francisco, CA</Location>
       <Name>GenComm</Name>
       <ListOfContact>
         <Contact>
           <FirstName>GENCOMM1</FirstName>
           <LastName>GENCOMM1</LastName>
           <MiddleName/>
           <Organization>Default Organization</Organization>
         </Contact>
         <Contact>
           <FirstName>GENCOMM2</FirstName>
           <LastName>GENCOMM2</LastName>
           <MiddleName/>
           <Organization>Default Organization</Organization>
         </Contact>
       </ListOfContact>
    </Account>
 </ListOfAccount>
</SiebelMessage>
```

Result Record in Database

The following code represents the results of the sync operation after acting on the two contacts from the database.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId = "1-2QY5" IntObjectName = "EAI Account" MessageType =
"Integration Object" IntObjectFormat = "Siebel Hierarchical">
 <ListOfAccount>
    <Account>
       <AccountStatus>Inactive</AccountStatus>
       <CurrencyCode>USD</CurrencyCode>
```

```
          <LanguageCode>ENU</LanguageCode>
          <Location>San Francisco, CA</Location>
          <Name>GenComm</Name>
          <ListOfContact>
            <Contact>
              <FirstName>GENCOMM1</FirstName>
              <LastName>GENCOMM1</LastName>
              <MiddleName/>
              <Organization>Default Organization</Organization>
            </Contact>
            <Contact>
              <FirstName>GENCOMM2</FirstName>
              <LastName>GENCOMM2</LastName>
              <MiddleName/>
              <Organization>Default Organization</Organization>
            </Contact>
          </ListOfContact>
      </Account>
   </ListOfAccount>
 </SiebelMessage>
```

In this case, if a matching Account1 exists in the database, then the EAI Siebel Adapter will perform an update of that record. If no record matching Account1 exists, then the EAI Siebel Adapter will insert a new account.

For all the matching child contacts, the sync operation is inherited. Therefore, if the child exists, it will be updated. If the child does not exist, 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 16 is a high level representation of a parent node using the update operation of the Execute method.

NOTE: The examples represented by Table 16, Table 17 on page 129, and Table 20 on page 130 basically have the same result. However, as reflected in Table 19 on page 130, the children do not automatically inherit *Update* if it is only set for the root.

Table 16. Representation of a Parent Node Using the Update Method

Record In Database	Integration Object Instance	Record After Execute Operation
Account1	Account1 operation=update	Account1
Contact0	Contact1	Contact1
Contact1	Contact2	Contact2

In this case, if a record matching Account1 exists in the database, then the EAI Siebel Adapter updates that specific record. If no matching account exists, then the result of the EAI Siebel Adapter is an error with this message:

```
Insert operation on integration component 'Account' failed because a matching record
in business component 'Account' with search specification '[Name] = "GenComm" AND
[Location] = "San Francisco, CA"' was found (SBL-EAI-04383)
```

For all the matching child contacts, the update operation is inherited. Therefore, if the child exists, it will be updated. If the child does not exist, it is inserted. For child contacts that exist in the database but do not match the integration object instance, they will be deleted. These may 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 17 is a high level representation of this example.

Table 17. Representation of Two Operations

Record In Database	Integration Object Instance	Record After Execute Operation
Account1	Account1 operation=update	Account1
Contact0	Contact1	Contact1
Contact1	Contact2 operation=insert	Contact2

In this case, if a record matching Account1 exists in the database, then the EAI Siebel Adapter updates that record. If no record matching Account1 exists, then the result from the EAI Siebel Adapter is an error.

You can also override the parent operation as in the case for Contact2. Since Contact2 does not exist, and there is an explicit insert operation, it will be inserted. Any unmatched children will be deleted as part of the parent operation (update). This is the reason why Contact0 is deleted.

If you are explicitly overriding the parent operation, you must make sure the operation applies. For example, the two combinations in Table 18 and Table 19 on page 130 will fail. In Table 18, it fails because an insert is attempted when Contact1 already exists in the database.

Table 18. Representation of Overriding a Parent Operation Using Insert

Record In Database	Integration Object Instance	Record After Execute Operation
Account1	Account1 operation=update	
Contact0	Contact1 operation=insert	
Contact1	Contact2 operation=insert	

In Table 19, the update fails since SubContact3 inherits from Contact2's operation, and Subcontact3 does not exist in the database.

Table 19. Representation of Overriding a Parent Operation Using Update

Record In Database	Integration Object Instance	Record After Execute Operation
Account1	Account1	
Contact1	Contact1	
Contact2	Contact2 operation=update	
SubContact1	SubContact1	
SubContact2	SubContact3	

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 20 is a high level representation of this example.

Table 20. Representation of Overriding a Parent Operation Using Upsert

Record In Database	Integration Object Instance	Record After Execute Operation
Account1	Account1 operation=update	Account1
Contact0	Contact1	Contact1
Contact1	Contact2 operation=upsert	Contact2

In this case, if a record matching Account1 exists in the database, then the EAI Siebel Adapter updates that record. If no record matching Account1 exists, then the result of the EAI Siebel Adapter is an error.

For a record matching Contact2, the upsert operation overrides the update operation. Therefore, if Contact2 exists, it is updated. If no record matching Contact2 is found, 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 21 is a high level representation of this example.

Table 21. Representation of Overriding a Parent Operation Using Sync

Record In Database	Integration Object Instance	Record After Execute Operation
Account1	Account1 operation=upsert	Account1
Contact0	Contact1	Contact0
Organization2	Organization1	Organization2
Contact1	Contact2 operation=sync	Contact1
Organization2	Organization3	Organization1
Contact2		Organization2
Organization2		Contact2
		Organization3

In this case, if a record matching Account1 exists in the database, then the EAI Siebel Adapter updates that record. If no record matching Account1 exists, then the EAI Siebel Adapter inserts the record.

For all child contacts, the upsert operation applies. Therefore, if the child exists, it is updated. If the child does not exist, it is inserted. For child contacts that exist in the database, but do not match the integration object instance, they will remain unchanged because upsert does not delete children.

In the case of Contact2, which has the sync operation overriding the upsert operation, it is updated, and its children are synchronized.

Skiptree Operation

The whole sub tree rooted at this node is not processed. It is the same as that whole sub tree not existing in the integration object instance. Operations specified in child nodes do not affect processing in any way since the EAI Siebel Adapter does not act on the child.

```
  <?xml version="1.0" encoding="UTF-8"?>
  <?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId="1-2RE" MessageType="Integration Object" IntObjectName="Sample
  Account" IntObjectFormat="Siebel Hierarchical">
   <ListOfSampleAccount>
      <Account operation="upsert">
         <Name>foo </Name>
         <Location>cold storage</Location>
         <ListOfContact>
           <Contact operation="skiptree">
             <FirstName>firstname</FirstName>
             <LastName>contact1</LastName>
             <Organization>Default Organization</Organization>
             <PersonalContact>N</PersonalContact>
             <ListOfBusinessAddress>
             <BusinessAddress operation="insert">
               <City>San Mateo</City>
               <Zip>94402</Zip>
               <AddressName>primary address</AddressName>
             </BusinessAddress>
             </ListOfBusinessAddress>
           </Contact>
           <Contact>
             <FirstName>firstname</FirstName>
             <LastName>contact2</LastName>
             <Organization>Default Organization</Organization>
             <PersonalContact>N</PersonalContact>
           </Contact>
         </ListOfContact>
      </Account>
   </ListOfSampleAccount>
  </SiebelMessage>
```

Based on this example, the account is upserted. The processing of the first contact is completely skipped although the business address child has an *insert* operation set. Also, the second contact is upserted.

If the *skiptree* operation is specified for the account integration component, then the EAI Siebel Adapter skips processing the complete account. This results in no operation. It is possible to have many accounts with some having *skiptree* specified as shown in the following example. The EAI Siebel Adapter processes the trees that do not have *skiptree* specified.

```
  <?xml version="1.0" encoding="UTF-8"?>
  <?Siebel-Property-Set EscapeNames="false"?>
  <SiebelMessage MessageId="1-2RE" MessageType="Integration Object"
  IntObjectName="Sample Account" IntObjectFormat="Siebel Hierarchical">
   <ListOfSampleAccount>
      <Account operation="skiptree">
         <Name>foo</Name>
         <Location>cold storage<Location/>
      </Account>
      <Account operation="upsert">
         <Name>bar</Name>
         <Location>cold storage<Location/>
```

```
    </Account>
  </ListOfSampleAccount>
</SiebelMessage>
```

Skipnode Operation

Similar to all other Execute operations, the children nodes inherit the semantics of the operation from the parent nodes. If a node has operation *skipnode* set, then the EAI Siebel Adapter will skip setting field values for all children unless a child has an explicit operation set that will override.

```
<?xml version="1.0" encoding="UTF-8"?>
<?Siebel-Property-Set EscapeNames="false"?>
<SiebelMessage MessageId="1-2RE" MessageType="Integration Object" IntObjectName=
"EAI Account"  IntObjectFormat="Siebel Hierarchical">
 <ListOfAccount>
    <Account operation="skipnode">
       <Name>foo</Name>
       <Location>cold storage</Location>
       <ListOfContact>
         <Contact operation="upsert">
       <IntegrationId>1-123</IntegrationId>
           <FirstName>firstname</FirstName>
           <LastName>contact1</LastName>
         <ListOfContact_Organization>
           <Contact_Organization>
             <Organization operation="insert">MyOrganization</Organization>
             </Contact_Organization>
       </ListOfContact_Organization>
         </Contact>
         <Contact operation="upsert">
       <IntegrationId>2-123</IntegrationId>
         <FirstName>firstname</FirstName>
         <LastName>contact2</LastName>
         </Contact>
       </ListOfContact>
    </Account>
 </ListOfAccount>
</SiebelMessage>
```

Based on this example, the account is skipped. However, the EAI Siebel Adapter will attempt to insert the two contacts.

EAI Siebel Adapter Method Arguments

Each of the EAI Siebel Adapter methods takes arguments that allow you to specify required and optional information to the adapter. You can locate the arguments for each method in Table 22.

Table 22. EAI Siebel Adapter Method Arguments

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

Table 23 presents each argument of the EAI Siebel Adapter methods.

Table 23. Defining EAI Siebel Adapter Method Arguments

Argument	Display Name	Description
IntObjectName	Integration Object Name	Name of the integration object to delete.
NumOutputObjects	Number of Output Integration Objects	Number of output integration objects.
OutputIntObjectName	Output Integration Object Name	The name of the integration object that is to be output.
PrimaryRowId	Object Id	The PrimaryRowId refers to the Id field in the Business Component, Row_Id at the table level. PrimaryRowId is only returned as an output argument if you are passing in one integration object instance. If you are passing multiple integration object instances, then this argument is not returned as an output argument. To obtain the ID field when multiple integration objects are processed, use the StatusObject argument.
QueryByUserKey	Query By Key	A Boolean argument. Forces the EAI Siebel Adapter to use only the user keys to perform a query.
DeleteByUserKey	Delete By User Key	A Boolean argument. Forces the EAI Siebel Adapter to use only the user keys to identify a record.
ErrorOnNonExistingDelete	Error On Non Existing Delete	A Boolean argument. Determines whether or not the EAI Siebel Adapter should abort the operation if no match is found.
SiebelMessage	Siebel Message	The input or the output integration object instance.
searchspec	Search Specification	This argument allows you to specify complex search specifications as free text in a single method argument. For information, see "About Using Language-Independent Code with the EAI Siebel Adapter" on page 138.
StatusObject	Status Object	This argument tells the EAI Siebel Adapter whether or not to return a status message.

Table 23. Defining EAI Siebel Adapter Method Arguments

Argument	Display Name	Description
MessageId	Message Id	The MessageId can be used to specify the ID for the generated message. By default, the EAI Siebel Adapter generates a unique ID for each message. However, if you want to use the workflow process instance ID, then you can use this argument to specify the ID.
BusObjCacheSize	Business Object Cache Size	Default is 5. Maximum number of Business Objects instances cached by the current instance of the EAI Siebel Adapter. If set to zero, then the EAI Siebel Adapter does not use the cache.
LastPage	Last Page	Boolean indicating whether or not the last record in the query result set has been returned.
NewQuery	New Query	Default is False. Boolean indicating whether a new query should be executed. If set to True, a new query is executed flushing the cache for that particular integration object.
PageSize	Page Size	Default is 10. Indicates the maximum number of integration object instances to be returned.
StartRowNum	Starting Row Number	Default is 0 (first page). Indicates the row in the result set for the QueryPage method to start retrieving a page of records.
ViewMode	View Mode	Default is All. Visibility mode to be applied to the Business Object. Valid values are: Manager, Sales Rep, Personal, Organization, Sub-Organization, Group, Catalog, and All. Note that the ViewMode user property on the integration object has priority over the ViewMode method argument.
SortSpec	Sort Specification	Default is the SortSpec of the underlying business component. This argument allows you to specify complex sort criteria as a free text in a single method argument, using any business component fields and standard Siebel sort syntax—for examples, see *Using Siebel Tools*.

About the SearchSpec Input Method Argument

The SearchSpec input method argument is applicable to the QueryPage, Query, Delete, and Execute methods. This method argument allows you to specify complex search specifications as free text in a single method argument. Expressions within a single integration component are restricted only by the Siebel Query Language supported by the Object Manager. Integration components and fields are referenced using the following notation:

[*IntCompName*.*IntCompFieldName*]

For example, given an integration object definition with two integration components, Account as the root component and Contact as the child component, the following search specification is allowed:

```
([Account.Site] LIKE "A*" OR [Account.Site] IS NULL) AND [Contact.PhoneNumber] IS NOT NULL
```

This search specification queries accounts that either have a site that starts with the character A, or do not have a site specified. In addition, for the queried accounts, it queries only those associated contacts that 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 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 MVGs in the EAI Siebel Adapter

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 only when the Auto Primary property of the underlying MVLink is specified as Selected or None. If Auto Primary is defined as Default, then the Object Manager does not allow the EAI Siebel Adapter to set the primary. The exception to this rule are all the visibility MVG components (components whose records are used by Object Manager to determine who is going to see their parent records). For information on Auto Primary property, see *Siebel Tools Online Help*.

Setting a Primary Position for a Contact

You have a contact with multiple contact positions in a Siebel application. None of these positions are marked as the primary position for the contact, and you want to select one of them as the primary position.

To specify a contact position as a primary

1 Create your XML file and insert <IsPrimaryMVG= 'Y'> before the contact position you want to identify as the primary position for the contact as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
   <?Siebel-Property-Set EscapeNames="false"?>
- <SiebelMessage MessageId="1-69A" IntObjectFormat="Siebel Hierarchical"
MessageType="Integration Object" IntObjectName="Sample Contact">
- <ListOfSampleContact>
- <Contact>
   <FirstName>Pal888</FirstName>
   <IntegrationId>65454398</IntegrationId>
   <JobTitle>Manager</JobTitle>
   <LastName>John888</LastName>
   <MiddleName />
   <PersonUId>1-Y88H</PersonUId>
   <PersonalContact>N</PersonalContact>
- <ListOfContact_Position>
- <Contact_Position IsPrimaryMVG="Y">
   <EmployeeFirstName>Siebel</EmployeeFirstName>
   <EmployeeLastName>Administrator</EmployeeLastName>
   <Position>Siebel Administrator</Position>
   <RowStatus>N</RowStatus>
   <SalesRep>SADMIN</SalesRep>
   </Contact_Position>
   </ListOfContact_Position>
   </Contact>
   </ListOfSampleContact>
   </SiebelMessage>.
```

2 Use the Upsert or Sync method to update the account.

About Using Language-Independent Code with the EAI Siebel Adapter

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

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

Multilingual LOV (MLOV) stores a language-independent code (LIC) in the Siebel Database that 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 should use LDV (N = no translation necessary) or LIC (Y = translation needed) for such LOVs.

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

Table 24 explains the behavior of the EAI Siebel Adapter according to the integration object user property *AllLangIndependentVals* values.

Table 24. EAI Siebel Adapter's Behavior for the User Property AllLangIndependentVals

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

Siebel EAI and Run-Time Events

The Siebel application allows triggering workflows based on run-time events or workflow policies.

Run-Time Events. Siebel EAI supports triggering workflows based on run-time events such as Write Record, which is triggered whenever a record is written. If you use the EAI Siebel Adapter to import data into Siebel Business Applications, and use run-time events, 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 invoked as soon as the EAI Siebel Adapter makes Write Record calls even though the EAI Siebel Adapter may be in the middle of a transaction. If you have export data workflows triggered on such events, this may lead to exporting data from Siebel 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 may 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 may 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 section. Use Workflow Policies instead of Run-Time Events when possible.

Best Practices for Using the EAI Siebel Adapter

The following best practices 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 need to 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.
- Siebel Systems 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

The EAI Siebel Adapter natively accesses Siebel objects definitions using the business objects, integration objects, and business component classes. Because of this design, you may get an EAI Siebel Adapter error that contains an error message from the Siebel Object Manager. See Figure 1 on page 14 for a logical overview of the Siebel architectural layers. Figure 1 on page 14 also shows the component events that will help you determine in which layer of the application the problem is occurring in.

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 the Siebel workflow process and business service. The workflow processes 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 may get an error in a step of the workflow process 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, you should 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 need to 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 EAI Siebel Adapter Logging

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 25 on page 142. These components events can be enabled on any server component that is capable of running an EAI process and on the Siebel client. You may wish to enable other events not listed in Table 25 on page 142.

For more information on how to perform logging in the Siebel clients, see *Siebel System Monitoring and Diagnostics Guide*.

Table 25. Component Events for Debugging EAI Siebel Adapter Problems

Event Alias Name	Logging Level	Description
EAISiebAdpt	4/5	Captures EAI Siebel Adapter related events, including integration component and integration component fields accessed and the values for the fields; business components and business component fields accessed and the values for the fields. This is the main event to enable for EAI Siebel Adapter troubleshooting.
EAISiebAdptPerf	4	Captures EAI Siebel Adapter performance related events, including operation performed and time for the operation in milliseconds. This event summarizes the result of the EAI Siebel Adapter operation. For more information on performance logging, see "Troubleshooting the EAI Siebel Adapter" on page 141 and Doc ID 476905.1 (a HOWTO document) on Oracle*MetaLink* 3. This document was previously published as Siebel FAQ 1840.
EAISiebAdptSvcArgTrc	3/4	Dumps the inputs and output arguments for the EAI Siebel Adapter when EnableServiceArgTracing=true. See "Enabling Siebel Argument Tracing" on page 143 for more information.
EAITransaction	4	Captures when an EAI Transaction starts.
EAIInfra	4	Output Message: IntObjType=Contact Interface Format=Siebel Hierarchical
EAIQrySpec	4	Captures the search specification if one is specified.
SQL	4	Captures SQL executed against the database.
SQLParseAndExecute	4	Captures SQL statements and shows SQL bind parameters executed. Shows SQL executed against the database. May sometimes be different than the SQL show in ObjMgrSQLLog.
ObjMgrLog	4/5	Logs error code and error message encountered by various Siebel objects.
ObjMgrDataLog	4	Logs the beginning of a transaction for the database connection.

Table 25. Component Events for Debugging EAI Siebel Adapter Problems

Event Alias Name	Logging Level	Description
ObjMgrBusServiceLog	4	Captures creation, deletion and invocation of a Business Service.
ObjMgrBusCompLog 4	4/5	Captures the beginning and end of the Business Component creation and deletion.

For all the events listed in Table 25, setting the logging level to level 4 is sufficient for most types of testing. You can set the component event to level 5 if you want to see debug level output, but it is not generally recommended as it adds more lines of data to the log file that may or may not be helpful. Logging level 4/5 represents that a logging level of 4 or 5 is supported.

To enable EAI Siebel Adapter logging

1. Navigate to the Administration - Server Configuration > Servers view.
2. In the top applet, select the Siebel Server that you want to enable EAI Siebel Adapter logging.
3. In the middle applet, select the Components tab, and highlight the component.
4. In the lower applet, select the Events tab, and set component events.

When you enable the component event logging, make sure you select the appropriate server component or components involved in the process. For example, if you are testing receiving XML data in the MQSeries Server Receiver, then you would enable logging on the MQSeriesSrvRcvr component.

You can also use the same srvrmgr command to turn on the component event logging. You will use the "%" shortcut syntax to enable events. An example of this syntax is

```
change evtloglvl EAISIEB%=4 for comp BusIntMgr
```

Enabling Siebel Argument Tracing

You can also export input and output arguments in XML format to a file for the EAI Siebel Adapter. These XML files represent the input and output arguments integration object instances. This is a useful technique as it writes to a file the integration object instances in the directory where your Siebel process is running. For example, in the Siebel Developer Web Client, it 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, add the following to your .cfg file:

     ```
     [EAISubsys]

     EnableServiceArgTracing = TRUE
     ```

   - If you are running the Siebel Web Client, modify the following Siebel Server parameter for your object manager:

```
"EnableServiceArgTracing" = true
```

2 Set the appropriate component event level on your server component through the server manager on the server or SIEBEL_LOG_EVENTS in the Siebel Developer Web Client.

If you set event to:

- =3, then input arguments will be written out to a file when an error happens.
- =4, then input and output arguments will be written to a file.

Configuring the EAI Siebel Adapter 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.

About the Modification Key

A Modification Key is an Integration Component Key of the type *Modification Key*. A Modification Key is a collection of fields that together should be used to verify the version of an integration component instance. Typically, Modification Key fields are Mod Id fields for the tables used. Multiple Modification Key fields may be needed, because a business component may be updating multiple tables, either as extension tables, or through implicit or explicit joins.

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, 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, the EAI Siebel Adapter assumes that the record has been deleted by other users and returns the error SSASqlErrWriteConflict.

If a valid Modification Key as well as the corresponding record can be found, the EAI Siebel Adapter checks if the Modification Key fields in the XML request and the matched record are consistent. If any of the fields are inconsistent, the EAI Siebel Adapter assumes that the record has been modified by other users and returns the error SSASqlErrWriteConflict. If all the fields are consistent, the EAI Siebel Adapter proceeds with the requested operation.

About Modification IDs

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

NOTE: The EAI Siebel Adapter can update base and extension tables. It may 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, you 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 should:

- Be prefixed with CX string and preferably followed by the name of the join
- Be Joined over the correct join
- Have MODIFICATION_NUM specified as underlying column of type DTYPE_INTEGER

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 for concurrency control

1. For each integration component, identify all needed Modification IDs:

   NOTE: In addition to the Modification ID for the base table, Modification IDs for tables that are used through one-to-one extension as well as through implicit joins are relevant. For example, on modifying an account record MODIFICATION_NUM column on S_ORG_EXT is updated, not the MODIFICATION_NUM column on S_PARTY.

   a. Identify all active fields in an integration component that will be updated and have to be concurrency safe.

   b. Select the corresponding business component, the value in the External Name property of the integration component.

c For each field identified in Step a, check the value of the Join property of the field. If the join is not specified, then the field belongs to the base table; otherwise, note the name of the join.

d In the Object explorer, select Business Component > Join and query for the business component from Step b. Search whether there is an entry whose Alias property matches the name of the join from Step c:

- If a matching Alias is found, then this field belongs to a Joined Table. The name of the join in Step c is the join name, and the value of the Table property is the joined table.
- If no Alias matches, then this is an implicit join to an Extension Table. The name of the join in Step c is the name of the extension table.

2 Create business component fields for Mod Ids of Joined Tables. For the above example, create a new field in 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 needs to be done.

5 Repeat the following steps for each Modification ID field on the integration component:

a Add user property MVGFieldName if MVG, or AssocFieldName if MVG Association.

b Set the value of the user property to the same as the field name—for example, Mod Id, *extension table name*.Mod Id, or CX_*join*.Mod Id.

6 Create Modification Key.

Define a new integration component key of type Modification Key, and include all the integration component fields exposed in Step 3 on page 146 to this key.

7 Validate integration objects and compile a new SRF.

8 Modify client program to use the Modification Key mechanism:

a The client program should store the value of the Modification IDs when it queries data from the Siebel Database.

b The client program should send exactly the same values of the Modification IDs that it retrieved from the Siebel Database when sending an update.

c The client program should not send any Modification IDs when sending a new record to the Siebel application. If this is violated, the client program generates an error indicating that the record has been deleted by another user.

Integration Component Account Example

Consider an integration component Account of the business component Account:

- Field Home Page has property Join set to S_ORG_EXT. This is an implicit join, because it is not listed in the joins; therefore, this field belongs to Extension Table S_ORG_EXT.
- Field Primary Organization has property Join set to Primary Organization-S_BU. This is an explicit join, 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.

About Status IDs

When using Status IDs with Modification IDs, the following behavior can be present:

- All fields must be present in the integration object instance for the Modification Key to be used.
- Only one defined Modification Key is present for each integration component. Unlike User Keys, multiple Modification Keys are not used with Status IDs.

7 EAI UI Data Adapter

This chapter includes the following topics:

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 26.

- **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 Row Id and Mod Id fields.

  For more information about how Status Keys are used with the EAI Siebel Adapter, see "About Integration Component Keys" on page 26.

- **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 30.

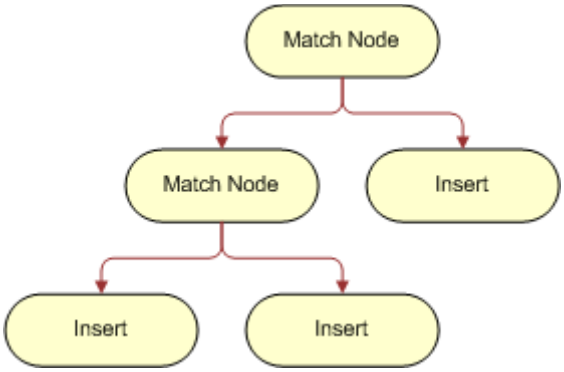


Figure 30. Operation Semantics on Leaf Nodes

The match nodes in Figure 30 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 151.

- **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 152.

- **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” on page 109.

EAI UI Data Adapter Methods

The EAI UI Data Adapter service provides access to the following methods:

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, 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 26 lists the method arguments used with the QueryPage method. For a description of the arguments, see "EAI UI Data Adapter Method Arguments" on page 166.

Table 26. Method Arguments for QueryPage

Method Argument Name	Input or Output
NewQuery	Input
OutputIntObjectName	Input
NumOutputObjects	Output
SiebelMessage	Input or Output
ViewMode	Input
NamedSearchSpec	Input
LOVLanguageMode	Input

Root and Child Pagination

The EAI UI Data Adapter supports pagination for both root and child components. To support root and child pagination, the EAI UI Data Adapter requires that you set the attributes listed in Table 27 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 invoke QueryPage.

Table 27. Attributes for Root and Child Pagination

Attribute	Description
pagesize	The number of records to be returned for a component. The default page size is 10. Note that there is a server parameter that controls the maximum page size (MaximumPageSize). If the pagesize attribute is greater than the maximum pagesize defined in the server parameter, an error occurs.
startrownum	Determines the starting point for record retrieval. The 0-based index of the record within the recordset.
lastpage	Indicates whether the record being returned is the last record in the record set. The value is set by the EAI UI Data Adapter. Valid values are true or false.
recordcountneeded	When set to true, indicates that a record count is needed for this component. Valid values are true or false.
recordcount	Value set by the EAI UI Data Adapter indicating the approximate record count provided by the object manager based on the search specification.

Example of QueryPage on Parent and Child Components

This example demonstrates querying on both parent and child components. In this example, the query is for accounts that begin with 'A' and any associated contacts (First Name and Last Name). Note that pagesize is 10 and an approximate record count is requested and returned in the response.

Request

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount pagesize="10" startrownum="0" recordcountneeded = "true">
    <Account>
      <Name>='A'</Name>
        <ListOfContact>
        <Contact>
          <FirstName></FirstName>
          <LastName></LastName>
        </Contact>
```

```
        </ListOfContact>
      </Account>
    </ListOfAccount>
  </SiebelMessage>
```

Response

```
  SiebelMessage MessageType="Integration Object" IntObjectName="Account"
  IntObjectFormat="Siebel Hierarchical">
    <ListOfAccount recordcount="2" lastpage="true">
      <Account>
        <Name>Adams Tech</Name>
        <ListOfContact lastpage="true">
        <Contact>
          <FirstName>Sally</FirstName>
          <LastName>Brown</LastName>
        </Contact>
          <Contact>
            <FirstName>Terry</FirstName>
            <LastName>Smith</LastName>
          </Contact>
        </ListOfContact>
      </Account>
      <Account>
        <Name>Aleph Inc.</Name>
          <ListOfContact lastpage="true">
          <Contact>
            <FirstName>Bill</FirstName>
            <LastName>Jones</LastName>
          <Contact>
          <Contact>
            <FirstName>Roland</FirstName>
            <LastName>Smith</LastName>
          </Contact>
        </ListOfContact>
     </Account>
    </ListOfAccount>
  </SiebelMessage>
```

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 28. If both attributes are not specified, the field is not considered as a sort field.

Table 28. Sort Specification Attributes

Attribute	Description
sortorder	Determines whether the sort order is ascending or descending. Valid values are ASC or DEC.
sortsequence	Determines the order in which the sort specification is applied. Valid values are integer numbers.

Example of Sort Specification

This example demonstrates using the QueryPage method with an ascending sort order.

Request

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Row_Id>2-1111</Row_Id>
      <ListOfContact pagesize="40" startrownum="0" recordcountneeded="true">
        <Contact>
          <FirstName sortorder="ASC" sortsequence="1"></FirstName>
        </Contact>
      </ListOfContact>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```

Response

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount lastpage="true">
    <Account>
      <Row_ID>2-1111</Row_ID>
      <ListOfContact recordcount="3" lastpage="true">
        <Contact>
          <FirstName>Alice</FirstName>
        </Contact>
        <Contact>
          <FirstName>Bill</FirstName>
        </Contact>
        <Contact>
          <FirstName>Casey</FirstName>
        </Contact>
```

```
        </ListOfContact>
      </Account>
    </ListOfAccount>
</SiebelMessage>
```

Predefined Query

You can specify the name of a PDQ using the method argument NamedSearchSpec. The EAI UI Data Adapter uses this value to set the search specification at the business object level.

Search Specification

You can use the searchspec attribute on a component instance for complicated queries.

For example, query by example (QBE) uses AND as the implicit operator between fields. You could implement OR semantics by using multiple integration component instances, but this would result in a query for each integration component instance and might result in duplicate records being returned. Using the searchspec attribute could avoid this problem.

The syntax for the searchspec attribute is as follows:

- Expression: *Expression* [*Binary Operator Expression*]
- Expression: [*Field XML tag*] *Operator* '*Value*'
- Expression: (*Expression*)

  NOTE: Parentheses can be nested.
- Expression: [*Field XML tag*] IS NULL | [*Field XML tag*] IS NOT NULL
- Expression: EXISTS(*Expression*) | NOT EXISTS(*Expression*)

  NOTE: In EXISTS and NOT EXISTS expressions, use the business component field names of multivalue group (MVG) fields, not the integration component XML tag names.
- Operator: = | ~= | < | <= | > | >= | <> | LIKE | ~LIKE
- Binary Operators: 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 29.

Table 29. Example Search Specification Conversion

This Search Spec	Will be converted to this
[FirstName] LIKE '*Jon*' AND [LastName] = 'Doe'	[First Name] LIKE '*Jon*' AND [Last Name] = 'Doe'
[FirstName] LIKE '*Jon*' OR [LastName] LIKE 'Doe*'	[First Name] LIKE '*Jon*' OR [Last Name] LIKE 'Doe'

- If the input argument LOVLanguageMode is set to LIC, converts LOV values to language dependent codes. See "EAI UI Data Adapter Method Arguments" on page 166.
- Validates operators, binary operators, and the syntax of the searchspec.

For more information about query language, see the *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 invoked 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 30 lists the method arguments used with UpdateLeaves. For a complete description of the method arguments, see "EAI UI Data Adapter Method Arguments" on page 166.

Table 30. Method Arguments for UpdateLeaves

Method Argument Name	Input or Output
BusinessObjectCacheSize	Input
SiebelMessage	Input or Output
LOVLanguageMode	Input

Example of Updating Root Component

The following example demonstrates updating a root component.

Request

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>1-1-1111</Account>
    <Employees>4900</Employees>
  </ListOfAccount>
</SiebelMessage>
```

Response

```
<SiebelMessage MessageId="P-3ITT" MessageType="Integration Object"
IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Id>1-1-1111</Id>
      <Mod_Id>2</Mod_Id>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```

Example of Updating Child Component

This example demonstrates updating a child component.

Request

```
<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

```
<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 invoked 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 31 lists the method arguments used with the InitLeaves Method. For a complete description of the method arguments, see "EAI UI Data Adapter Method Arguments" on page 166.

Table 31. Method Arguments for InitLeaves

Method Argument	Input or Output
BusinessObjectCacheSize	Input
SiebelMessage	Input or Output
LOVLanguageMode	Input
ViewMode	Input

Example of Using InitLeaves on a Root Component

The following code snippet demonstrates using InitLeaves to retrieve default values for a root component. In this example the root component is Account.

Request

```
<?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

```
<?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

```
<?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

```
<?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 invoked 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, 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 149.
- **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 should 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 may or may 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 32 lists the method arguments used with the InsertLeaves method. For descriptions of the methods, see "EAI UI Data Adapter Method Arguments" on page 166.

Table 32. Method Arguments for InsertLeaves

Method Argument Name	Input or Output
BusinessObjectCacheSize	Input
SiebelMessage	Input or Output
LOVLanguageMode	Input

Example of Inserting a Root Component

This example code snippet demonstrates inserting a non-existing account.

Request

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Type>Competitor</Type>
      <Name>Dixon Inc.</Name>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```

Response

```
<SiebelMessage MessageId="P-3ITI" MessageType="Integration Object"
IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Id>P-5NA84</Id>
      <Mod_Id>0</Mod_Id>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```

Example of Inserting a Child Component

The code snippets in this example demonstrate inserting a non-existing business address for an existing account.

Request

```
<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

```
<SiebelMessage MessageId="P-3ITJ" MessageType="Integration Object"
IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Id>P-5NA84</Id>
      <Mod_Id>1</Mod_Id>
      <ListOfBusiness_Address>
        <Business_Address>
          <Id>P-5NA8B</Id>
          <Mod_Id>0</Mod_Id>
        </Business_Address>
      </ListOfBusiness_Address>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```

Example of Inserting an Association Child Component

This example demonstrate inserting an existing organization for an existing account. This operation associates the organization with the account. If the organization does not exist, the EAI UI Data Adapter generates an error.

Request

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Id>P-5NA84</Id>
        <ListOfAccount_Organization>
        <Account_Organization>
          <Id>1-123</Id>
        </Account_Organization>
      </ListOfAccount_Organization>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```

Response

```
<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 33 lists the method arguments used with DeleteLeaves. For descriptions of the arguments, see "EAI UI Data Adapter Method Arguments" on page 166.

Table 33. Method Arguments for DeleteLeaves

Method Argument Name	Input or Output
IntObjectName	Input
SiebelMessage	Input or Output
LOVLanguageMode	Input

Example of Deleting a Root Component

This example demonstrates deleting a root component.

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Id>P-5NA84</Id>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```

Example of Deleting a Child Component

This example demonstrates deleting a child component.

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
```

```
        <Account>
          <Id>P-5NA84</Id>
          <ListOfBusiness_Address>
            <Business_Address>
            <Id>P-5NA8B</Id>
            </Business_Address>
          </ListOfBusiness_Address>
        </Account>
      </ListOfAccount>
    </SiebelMessage>
```

Execute Method

The Execute method allows you to perform multiple operations on multiple business components. It is the only method that operates on internal nodes. The Execute method returns the same kind of object that the InsertLeaves method returns. For more information, see “InsertLeaves Method” on page 160.

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 34:

Table 34. Operation Attributes for Execute Method

Operation	Description
update	Updates the integration component instance
insert	Inserts the integration component instance
delete	Deletes the integration component instance
skipnode	Matches integration component instances and process children

Operations must be specified on every integration component instance. If an operation is not specified, the Execute method returns an error.

Execute Method Arguments

Table 35 lists the method arguments used with the Execute method. For a description of the methods, see "EAI UI Data Adapter Method Arguments" on page 166.

Table 35. Method Arguments for Execute

Method Argument Name	Input or Output
BusinessObjectCacheSize	Input
SiebelMessage	Input or Output
LOVLanguageMode	Input

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

```
<SiebelMessage MessageType="Integration Object" IntObjectName="Account"
IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account operation="skipnode">
      <Id>1-1-1111</Id>
      <ListOfBusiness_Address>
        <Business_Address operation="update">
          <Id>2-2-2222</Id>
          <Postal_Code>94402</Postal_Code> <!--Postal Code changed-->
        </Business_Address>
        <Business_Address operation="insert">
          <Postal_Code>94402</Postal_Code>
          <City>San Mateo</City>
          <Street_Address>2215 Bridgepointe Parkway</Street_Address>
          <State>CA</State>
          <Country>USA</Country>
        </Business_Address>
      </ListOfBusiness_Address>
      <ListOfContact>
        <Contact operation="delete">
          <Id>4-4-4444</Id>
        </Contact>
      </ListOfContact>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```

Response

```
<SiebelMessage MessageId="42-21YQ" MessageType="Integration Object"
IntObjectName="Account" IntObjectFormat="Siebel Hierarchical">
  <ListOfAccount>
    <Account>
      <Id>1-1-1111</Id>
      <Mod_Id>3</Mod_Id>
      <ListOfBusiness_Address>
        <Business_Address>
          <Id>2-2-2222</Id>
          <Mod_Id>1</Mod_Id>
        </Business_Address>
        <Business_Address>
          <Id>42-53Q2W</Id>
          <Mod_Id>0</Mod_Id>
        </Business_Address>
      </ListOfBusiness_Address>
    </Account>
  </ListOfAccount>
</SiebelMessage>
```

EAI UI Data Adapter Method Arguments

The methods exposed in the EAI UI Data Adapter take arguments that you use to specify information that the adapter uses when processing the request. Table 36 summarizes the EAI UI Data Adapter method arguments. Subsequent sections describe each method, list the method arguments applicable to each method, and indicate whether the method argument is used for input or output.

Table 36. EAI UI Data Adapter Method Arguments

Argument	Display Name	Description
BusinessObjectCacheSize	Business Object Cache Size	Maximum Number of Business Objects that can be cached at one time.
LOVLanguageMode	LOV Language Mode	Indicates whether the EAI UI Data Adapter needs to 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, the EAI UI Data Adapter does not translate the value before sending it to the object manager.

Table 36. EAI UI Data Adapter Method Arguments

Argument	Display Name	Description
NamedSearchSpec	Predefined Query	Name of a PDQ. The EAI UI Data Adapter sets the name of the PDQ on the business object instance. If NamedSearchSpec and QBE are specified, NamedSearchSpec is used.
NewQuery	New Query	Default is False. Boolean indicating whether a new query should be executed. If set to True, a new query is executed flushing the cache for that particular integration object.
NumOutputObjects	Number of Output Integration Objects	Number of Integration Objects output
OutputIntObjectName	Not applicable	The name of the integration object that will be sent in the output.
SiebelMessage	Siebel Message	Input or output integration object instance.
ViewMode	View Mode	Visibility algorithm used in addition to a search specification to determine which records will be retrieved. The ViewMode method argument is used to set the View Mode property for all business components corresponding to the integration object. Valid values are *Manager*, *Sales Rep*, *Personal*, *Organization*, *Sub-Organization*, *Group*, *Catalog*, and *All.*

8 Siebel Virtual Business Components

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

A 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.

Figure 31 shows two approaches to building VBCs.

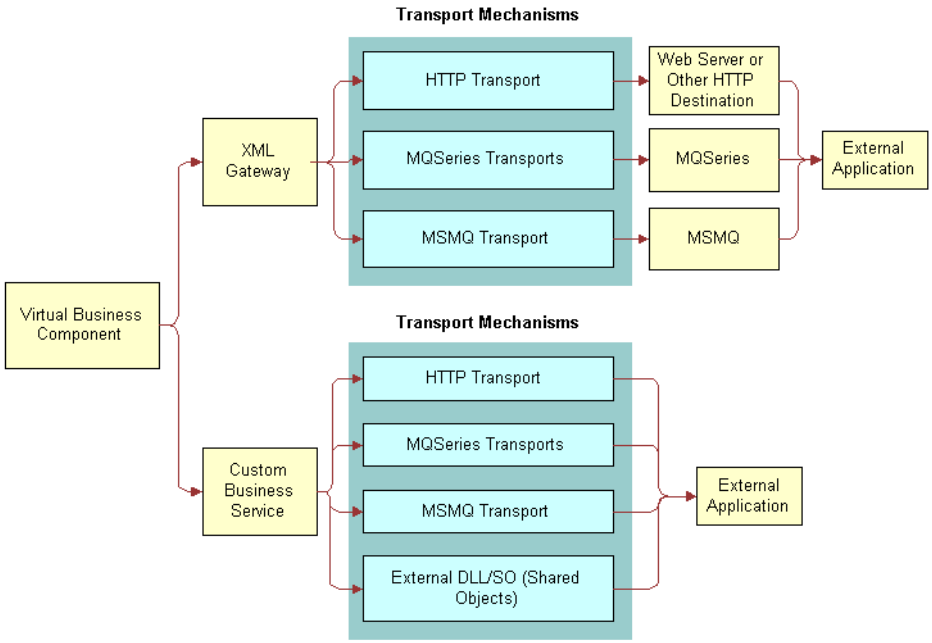


Figure 31. Two Approaches to Building Virtual Business Components

You can take two approaches to use VBCs, as illustrated in Figure 31:

- 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.

About 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 drilling down from a VBC. You can drill down on a VBC from a standard BC, 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 37 on page 172.

- You can pass search and sort specifications to the business service used by a VBC.
- You can use Validation, Pre Default Value, Post Default Value, Link Specification, and No Copy attributes of the VBC fields.
- You can use predefined queries with VBC.
- You can have picklists based on VBC, and use the picklist properties such as No Insert, No Delete, No Update, No Merge, Search Specification, and Sort Specification.
- You can use the Cascade Delete, Search Spec, Sort Spec, No Insert, No Update, and No Delete link properties when a VBC is the child business component on the link.
- You can use No Insert, No Update, No Delete, Search Spec, Sort Spec, and Maximum Cursor Size business component properties.

Usage and Restrictions of 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.
- Using the same name for the VBC field names and the remote data source field names may reduce the amount of required programming. (Optional)
- VBCs cannot be docked, so they do not apply to remote users.
- VBCs cannot contain a multi-value group (MVG).
- VBCs do not support many-to-many relationships.
- VBCs cannot be loaded using Enterprise Integration Manager.
- Standard business components cannot contain multi-value group 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.

Using Virtual Business Components

To use VBCs to share data with an external application, perform the following high-level tasks:

- Create a new VBC.

  For information, see “Creating a New Virtual Business Component.”

- Set the User Properties on VBCs.

  For information, see “Setting User Properties for the Virtual Business Component” on page 172.

- Configure your VBC Business Service:
  - Configure your XML Gateway Service or write your own Business Service.
  - For information, see "XML Gateway Service" on page 174 and "Custom Business Service Methods" on page 184.
  - Configure your external application.

  For information, see "External Application Setup" on page 184.

Creating a New Virtual Business Component

You create a new VBC in Siebel Tools.

To create a new virtual business component

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

Setting User Properties for the Virtual Business Component

When defining the VBC, you must provide the user properties shown in Table 37.

Table 37. Setting Virtual Business Component User Properties

User Property	Description
Service Name	The name of the business service.
Service Parameters	(Optional) Any parameters required by the business service. The Siebel application passes this user property, as an input argument, to the business service.

Table 37. Setting Virtual Business Component User Properties

User Property	Description
Remote Source	(Optional) External data source that the business service is to use. This property allows the VBC to pass a root property argument to the underlying business service, but it does not allow a connection directly to the external datasource. The Siebel application passes only this user property as an input argument.
VBC Compatibility Mode	(Optional) Determines the format of the property set passed from a VBC to a business service, or the format in which the outgoing XML from the XML Gateway will be. A valid value is Siebel *xxx*, where *xxx* can be any Siebel release number. Some examples would be Siebel 6 or Siebel 7.0.4. If *xxx* is less than 7.5, the format will be in a release that is earlier than release 7.5. Otherwise, a new property set, and the XML format will be passed. If you are creating a VBC in 7.5, there is no need to define this new user property, because the default is to use the new PropertySet from a VBC and the new outgoing XML from the XML Gateway. For your existing VBC implementation, update your VBC definition by adding this new user property, and setting it to Siebel *xxx*, where *xxx* is the version number that you want.

To define user properties

1. Start Siebel Tools.
2. Lock the appropriate project.
3. Click the Business Component folder in the Object Explorer to expand the hierarchical tree.
4. Select the business component you want to define user properties for.
5. Click the Business Component User Prop folder in the Object Explorer.
6. Choose Edit New Record to create a new blank user property record.
7. Type the name of the user property, such as Service Name, in the Name field.
8. Type the value of the user property, such as a business service name, in the Value field.
9. Repeat the process for every user property you want to define for this VBC.

NOTE: For the list of different property sets and their format, see "Examples of the Outgoing XML Format" on page 176 and "Examples of Incoming XML Format" on page 181.

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 176 and "Examples of Incoming XML Format" on page 181. 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 38. When using the XML Gateway, specify the following user properties for your VBC.

Table 38. User Properties

Name	Value
Service Name	XML Gateway
Service Parameters	*variable1 name=variable1 value*; *variable2 name=variable2 value*;...
Remote Source	*External Data Source*
VBC Compatibility Mode	Siebel *xxx*, where *xxx* can be any Siebel release number.

NOTE: You can concatenate multiple name-value pairs using a semicolon (;), but do not use any spaces between the name, the equal sign, the value, and the semicolon.

For example, if you want to specify the EAI HTTP Transport, you may use something like the following:

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

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

To implement 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 process to run, with the following user properties on the VBC:
   - **Service Name.** XML Gateway
   - **Service Parameters.** Transport=EAI Business Integration Manager (Server Request);ProcessName=EAITEST

3 Define a workflow process, EAITEST, to call the EAI MQSeries Server Transport with the SendReceive method.

4 Define a new process property, <Value>, on the workflow process, and use it as an output argument on the EAI MQSeries Server Transport step in the workflow process.

XML Gateway Methods

The XML Gateway provides the methods presented in Table 39.

Table 39. XML Gateway Methods

Method	Description
Init	Initializes the XML Gateway business service for every business component.
Delete	Deletes a given record in the remote data source.
Insert	Inserts a record into a remote data source.
PreInsert	Performs an operation that tests for the existence of the given business component. Only default values are returned from the external application.
Query	Queries the given business component from the given data source.
Update	Updates a record in the remote data source.

XML Gateway Method Arguments

The XML Gateway init, delete, insert, preInsert, query, and update methods take the arguments presented in Table 40.

Table 40. XML Gateway Arguments

Argument	Description
Remote Source	The VBC Remote Source user property. The remote source from which the service is to retrieve data for the business component. This must be a valid connect string. When configuring the repository business component on top of the specialized business component class CSSBCVExten, 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, it is passed to every request as the <remote-source> tag.
Business Component Id	Unique key for the given business component.
Business Component Name	Name of the business component or its equivalent, such as a table name.
Parameters	The VBC Service Parameters user property. A set of string parameters required for initializing the XML Gateway.

Examples of the Outgoing XML Format

Examples of the XML documents generated and sent by the XML Gateway to the external system are presented in Table 41 on page 177. These examples are based on the example in "Custom Business Service Example" on page 200. 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 41. Outgoing XML Tags and Descriptions

Method	Format of the XML Stream	Description
Delete Request	<siebel-xmlext-delete-req> <buscomp id="1">Contact</buscomp> <remote-source>http://throth/servlet/VBCContacts</remote-source> <row> <value field="AccountId">146</value> <value field="Name">Max Adams</value> <value field="Phone">(408)234-1029</value> <value field="Location">San Jose</value> <value field="AccessId">146</value> </row> </siebel-xmlext-delete-req>	siebel-xmlext-delete-req This tag requests removal of a single record in the remote system.
Init Request	<siebel-xmlext-fields-req> <buscomp id="1">Contact</buscomp> <remote-source>http://throth/servlet/VBCContacts</remote-source> </siebel-xmlext-fields-req>	■ siebel-xmlext-fields-req This tag fetches the list of fields supported by this instance. ■ buscomp Id The business component ID. ■ remote-source The remote source from which the service is to retrieve data for the business component.

Table 41. Outgoing XML Tags and Descriptions

Method	Format of the XML Stream	Description
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 41. Outgoing XML Tags and Descriptions

Method	Format of the XML Stream	Description
Query Request	<siebel-xmlext-query-req> <buscomp id="1">Contact</buscomp> <remote-source>http://throth/servlet/VBCContacts</remote-source> <max-rows>6</max-rows> <search-string>=([Phone] IS NOT NULL) AND ([AccountId] = "1-6")</search-string> <search-spec> <node node-type="BinaryOperator">AND <node node-type="UnaryOperator">IS NOT NULL <node node-type="Identifier">Phone</node> </node> <node node-type="BinaryOperator">= <node node-type="Identifier">AccountId</node> <node value-type="TEXT" node-type="Constant">1-6</node> </node> </node> </search-spec> <sort-spec> <sort field="Location">ASCENDING</sort> <sort field="Name">DESCENDING</sort> </sort-spec> </Siebel-xmlext-query-req>	■ siebel-xmlext-query-req This tag queries by example. The query-req XML stream contains parameters necessary to set up the query. In this example, the query requests that record information be returned from the remote system. ■ max-rows Maximum number of rows to be returned. The value is the Maximum Cursor Size defined at the VBC plus one. If the Maximum Cursor Size property is not defined at the VBC, then the max-rows property is not passed. ■ search-string The search specification used to query and filter the information. ■ search-spec Hierarchical representation of the search-string. For information, see “Search-Spec Node-Type Values” on page 180. ■ sort-spec List of sort fields and sort order.

Table 41. Outgoing XML Tags and Descriptions

Method	Format of the XML Stream	Description
Update Request	<siebel-xmlext-update-req> <buscomp id="2">Contact</buscomp> <remote-source>http://throth/servlet/VBCContacts</remote-source> <row> <value changed="false" field="AccountId">1-6</value> <value changed="false" field="Name">Max Adams</value> <value changed="true" field="Phone">(408)234-1029</value> <value changed="true" field="Location">San Jose</value> <value changed="false" field="AccessId">146</value> </row> </siebel-xmlext-update-req>	siebel-xmlext-update-req This tag requests changes to the field values for an existing row. All values for the record are passed with the <value> tags, and with the changed attribute identifying the ones that have been changed through the Siebel application.

Search-Spec Node-Type Values

The search-string is in the Siebel query language format. The search-string is parsed by the Siebel query object and then turned into the hierarchical search-spec. Table 42 shows the different search-spec node-types and their values.

Table 42. Search-Spec Node-Types

node-type	PropertySet/XML Representation
Constant	Example: <node node-type = "Constant" value-type="NUMBER">1000</node> The valid value-types are TEXT, NUMBER, DATETIME, UTCDATETIME, DATE, and TIME.
Identifier	Example: <node node-type="Identifier">Name</node> The value Name is a valid business component field name.
Unary Operator	Example: <node node-type="Unary Operator">NOT</node> The valid values are NOT, EXISTS, IS NULL, IS NOT NULL.
Binary Operator	Example: <node node-type= "Binary Operator" >AND</node> The valid values are LIKE, NOT LIKE, SOUNDSLIKE, =, <>, <=, <, >=, >, AND, OR, +, -, *, /, ^.

Examples of Incoming XML Format

Table 43 contains examples of XML documents that are sent from an external system to the XML Gateway in response to a request. These examples are based on the example in "Custom Business Service Example" on page 200. See Appendix C, "DTDs for XML Gateway Business Service," for examples of the DTDs that correspond to each of these methods.

Table 43. Incoming XML Tags and Descriptions

Method	Format of the XML Stream	Description
Delete Return	<siebel-xmlext-delete-ret />	siebel-xmlext-delete-ret. Only the XML stream tag is returned.
Error	<siebel-xmlext-status> <status-code>4</code> <error-field>Name</error-field> <error-text>Name must not be empty</error-text> </siebel-xmlext-status>	Format of the XML stream expected by the Siebel application in case of an error in the external application. If the error is specific to a field, the field name should be specified. The tags for this XML stream, and the entire XML stream, are optional: ■ siebel-xmlext-status This tag is used to check the status returned by the external system. ■ status-code This tag overrides the return value. ■ error-text This tag specifies textual representation of the error, if it is available. This tag appears in addition to the standard error message. For example, if the Siebel application attempts to update a record in the external system with a NULL Name, and this is not allowed in the external system, then the error text is set to: "Name must not be empty."

Table 43. Incoming XML Tags and Descriptions

Method	Format of the XML Stream	Description
Init Return	<siebel-xmlext-fields-ret> <support field="AccountId"/> <support field="Name"/> <support field="Phone"/> <support field="Location"/> <support field="AccessId"/> </siebel-xmlext-fields-ret>	siebel-xmlext-fields-ret The fields-ret XML stream return contains the list of VBC fields supported by the external application for this instance. The following field names are reserved by the Siebel application, and should not appear in this list: ■ Id ■ Created ■ Created By ■ Updated ■ Updated By
Insert Return	<siebel-xmlext-insert-ret> <row> <value field="AccountId">1-6</value> <value field="Name">Max Adams</value> <value field="Phone">(398)765-1290</value> <value field="Location">Troy</value> <value field="AccessId">146</value> </row> </siebel-xmlext-insert-ret>	siebel-xmlext-insert-ret If the remote system has inserted records, they can be returned to be reflected in the business component in an insert-ret XML stream in the <row> tag format as the insert-ret stream.
PreInsert Return	<siebel-xmlext-preinsert-ret> <row> <value field="Location">San Jose</value> </row> </siebel-xmlext-preinsert-ret>	siebel-xmlext-preinsert-ret Returns default values for each field, if there is any default value.

Table 43. Incoming XML Tags and Descriptions

Method	Format of the XML Stream	Description
Query Return	<siebel-xmlext-query-ret> <row> <value field="AccountId">1-6</value> <value field="Name">Sara Chen</value> <value field="Phone">(415)298-7890</value> <value field="Location">San Francisco</value> <value field="AccessId">128</value> </row> <row> <value field="AccountId">1-6</value> <value field="Name">Eric Brown</value> <value field="Phone">(650)123-1000</value> <value field="Location">Palo Alto</value> <value field="AccessId">129</value> </row> </siebel-xmlext-query-ret>	■ siebel-xmlext-query-ret The query-ret XML stream contains the result set that matches the criteria of the query. ■ row This tag indicates the number of rows returned by the query. Each row must contain one or more <values>. 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, it appears in the result set. If not, a unique key is generated. It is necessary to identify specific rows for DML operations. ■ value This tag specifies the field and value pairs and should be the same for each row in the set.

Table 43. Incoming XML Tags and Descriptions

Method	Format of the XML Stream	Description
Update Return	<siebel-xmlext-update-ret> <row> <value field="Location">San Jose</value> <value field="Phone">(408)234-1029</value> </row> </siebel-xmlext-update-ret>	siebel-xmlext-update-ret If the remote system updated fields, the fields can be returned to be reflected in the business component in an update-ret XML stream in the <row> tag format as the update-ret stream.

External Application Setup

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

Custom Business Service Methods

Your business service must implement the Init and Query methods as described in this section. The Delete, PreInsert, Insert, and Update methods are optional, and 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, 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.”

Common Method Parameters

Table 44 shows the input parameters common to every method. Note that all these parameters are at the root property set.

Table 44. Common Input Parameters

Parameter	Description
Remote Source	(Optional) Specifies the name of an external data source. This is the VBC's Remote Source user property, if defined. For information, see Table 37 on page 172.
Business Component Name	Name of the active VBC.
Business Component Id	Internally generated unique value that represents the VBC.
Parameters	(Optional) The VBC's Service Parameters user property, if defined. For information, see Table 37 on page 172. A set of parameters required by the business service.
VBC Compatibility Mode	(Optional) This is the VBC's Compatibility Mode user property, if defined. For information, see Table 37 on page 172.

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 Example" on page 200.

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

Delete

The Delete method is called when a record is deleted. Figure 32 illustrates the property set for the Delete input.

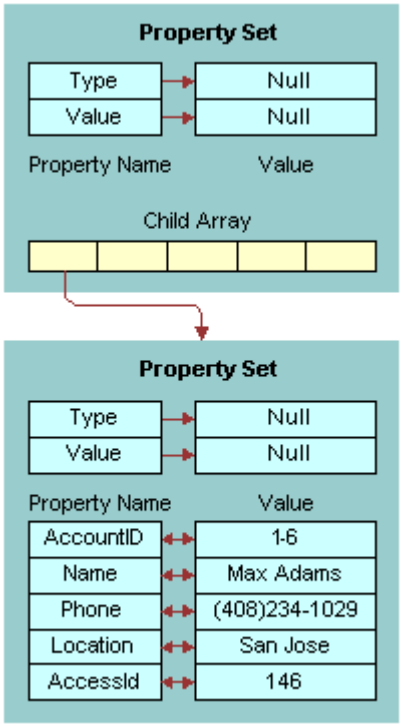


Figure 32. Delete Input Property Set

The following is the XML representation of the property set shown in Figure 32:

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

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

<PropertySet

    Business_spcComponent_spcId="1"

    Business_spcComponent_spcName="Contact">

    <PropertySet

    AccountId="1-6"

    Name="Max Adams"

    Phone="(408)234-1029"

    Location="San Jose"
```

```
      AccessId="146" />
  </PropertySet>
```

Figure 33 illustrates the property set for Delete output.

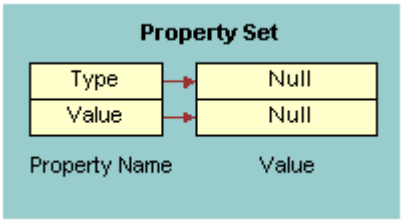


Figure 33. Delete Output Property Set

The following is the XML representation of the property set shown in Figure 33:

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

Error Return

Figure 34 illustrates the property set for the Error Return, when an error is detected.

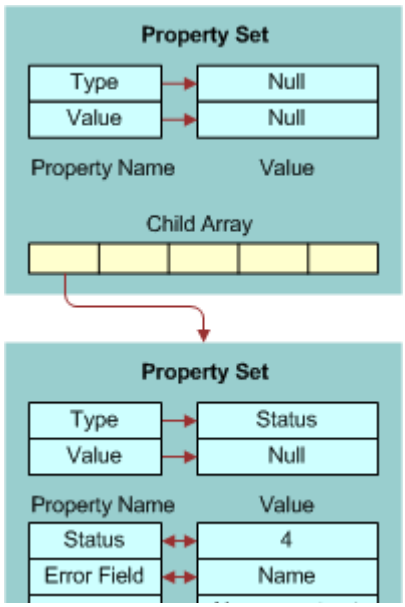


Figure 34. Error Return Property Set

The following is the XML representation of the property set shown in Figure 34 on page 187:

```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet>
   <Status Status="4"
      Error_spcField="Name"
      Error_spcText="Name must not be empty"/>
</PropertySet>
```

Init

The Init method is called when the 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 35 illustrates the property set for Init input.

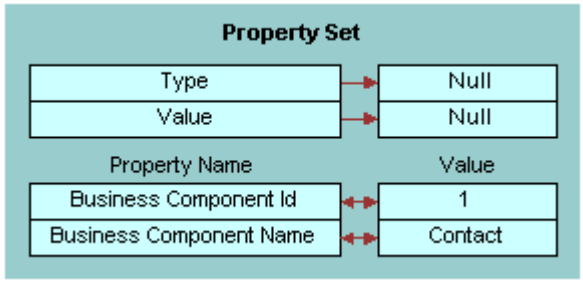


Figure 35. Init Input Property Set

The following is the XML representation of the property set shown in Figure 35:

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

Figure 36 illustrates the property set for Init output.

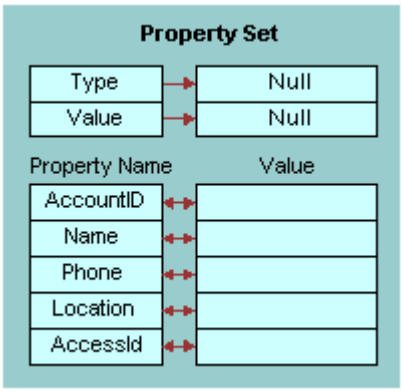


Figure 36. Init Output Property Set

The following is the XML representation of the property set shown in Figure 36:

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

Insert

The Insert method is called when a New Record is committed. Figure 37 illustrates the property set for Insert input.

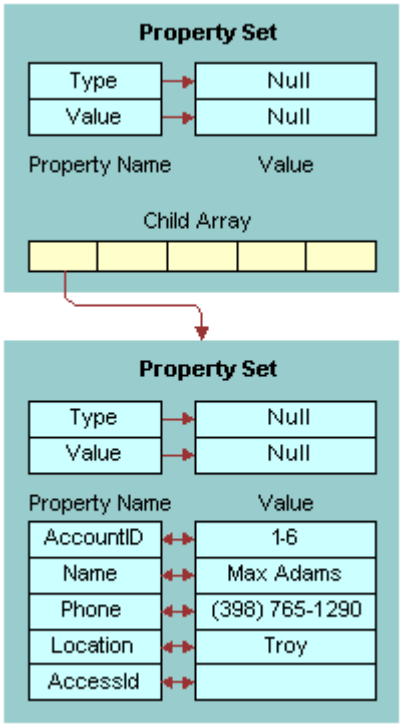


Figure 37. Insert Input Property Set

The following is the XML representation of the property set shown in Figure 37:

```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet
    Business_spcComponent_spcId="1"
    Business_spcComponent_spcName="Contact">
    <PropertySet
        AccountId="1-6"
        Name="Max Adams"
        Phone="(398)765-1290"
        Location="Troy"
```

```
        AccessId="" />
</PropertySet>
```

Figure 38 illustrates the property set for Insert output.

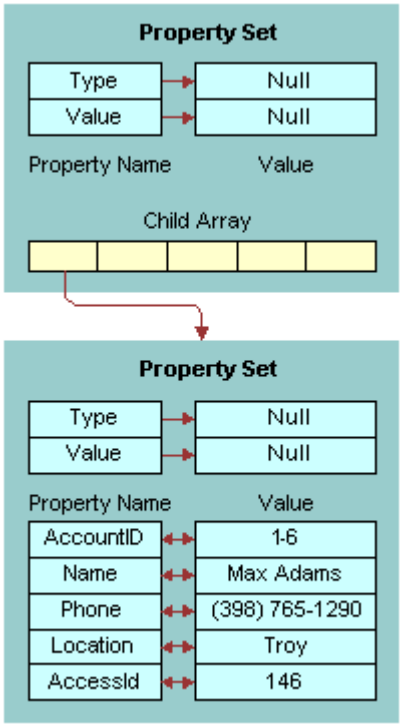


Figure 38. Insert Output Property Set

The following is the XML representation of the property set shown in Figure 38:

```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet
   <PropertySet
      AccountId="1-6"
      Name="Max Adams"
      Phone="(398)765-1290"
      Location="Troy"
      AccessId="146" />
</PropertySet>
```

PreInsert

The PreInsert method is called when a New Record operation is performed. It supplies default values. Figure 39 illustrates the property set for PreInsert input.

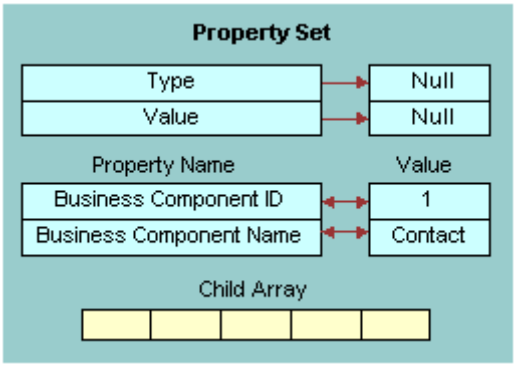


Figure 39. PreInsert Input Property Set

The following is the XML representation of the property set shown in Figure 39:

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

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

<PropertySet

    Business_spcComponent_spcId="1"

    Business_spcComponent_spcName="Contact"/>
```

Figure 40 illustrates the property set for PreInsert output.

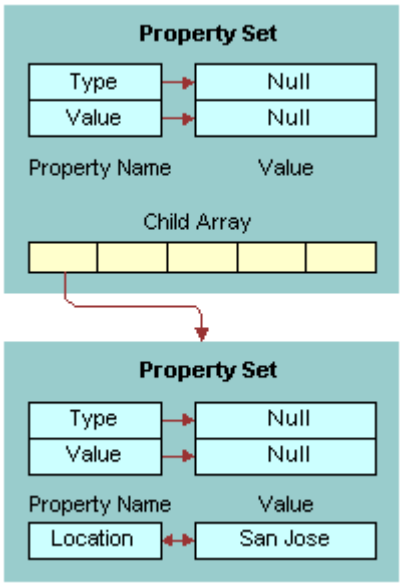


Figure 40. PreInsert Output Property Set

The following is the XML representation of the property set shown in Figure 40:

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

Query

The Query method is called when a search is performed. The Query method must be supported by every VBC. Each record that matches the query is represented as a property set. For example, if 5 records match the query, there will be 5 child property sets. Each property set contains a list of field names—field value pairs representing the values of each field for that particular record. Figure 41 and Figure 42 illustrate the property set for Query input.

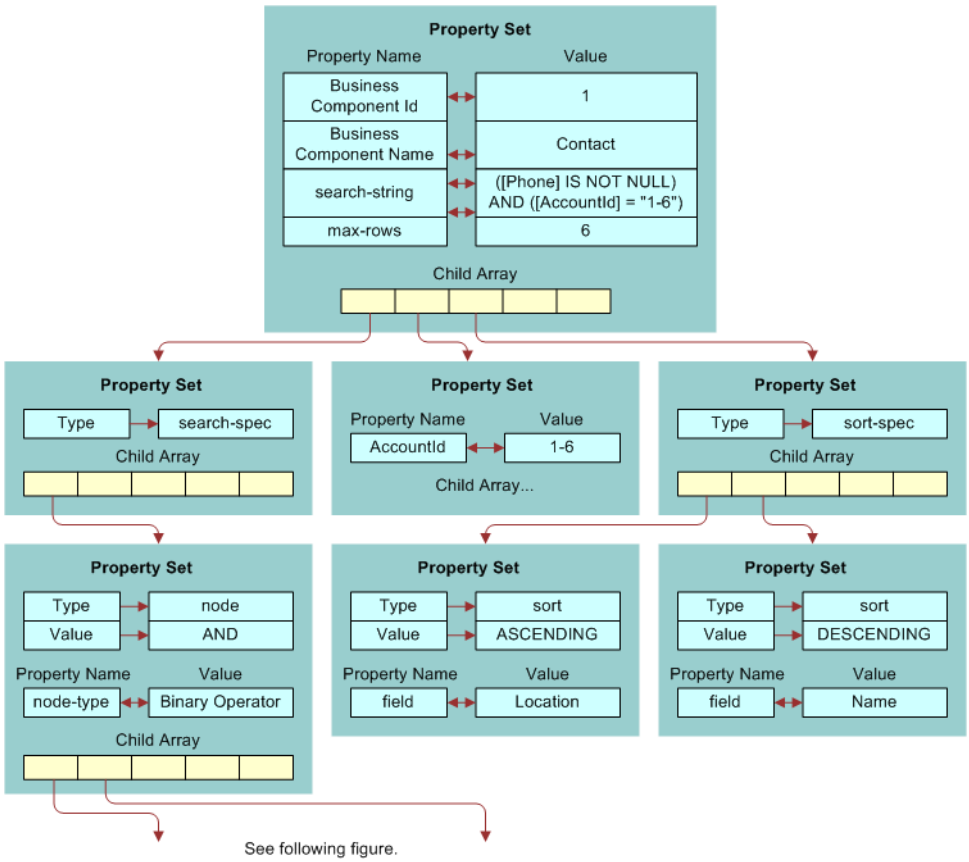


Figure 41. Query Input Property Set (Part 1)

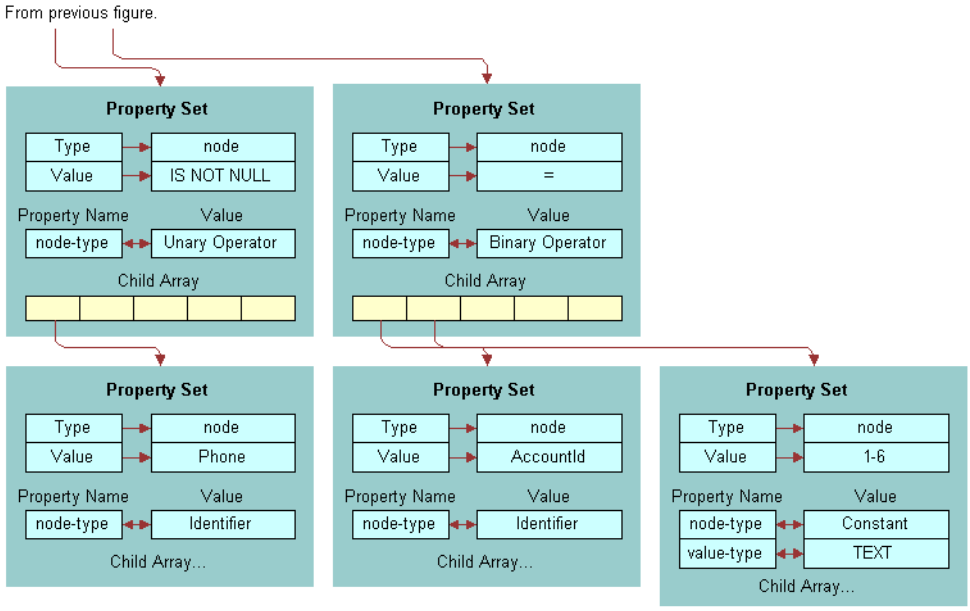


Figure 42. Query Input Property Set (Part 2)

The following is the XML representation of the property set shown in Figure 41 on page 194 and Figure 42:

```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet
   max-rows="6"
   search-string="([Phone] IS NOT NULL) AND ([AccountId] = "1-6")"
   Business_spcComponent_spcId="1"
   Business_spcComponent_spcName="Contact">
   <PropertySet AccountId="1-6" />
      <search-spec>
        <node node-type="Binary Operator">AND
        <node node-type="Unary Operator">IS NOT NULL
        <node node-type="Identifier">Phone</node>
          </node>
          <node node-type="Binary Operator">=
```

```
            <node node-type="Identifier">AccountId</node>
            <node value-type="TEXT" node-type="Constant">1-6</node>
            </node>
          </node>
        </search-spec>
        <sort-spec>
          <sort field="Location">ASCENDING</sort>
          <sort field="Name">DESCENDING</sort>
        </sort-spec>
    </PropertySet>
```

Figure 43 illustrates the property set for Query output.

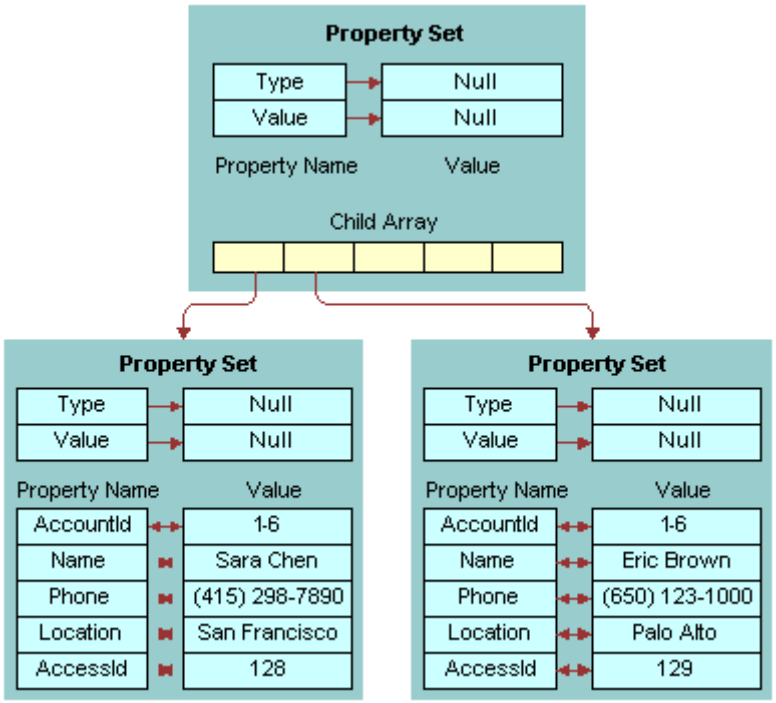


Figure 43. Query Output Property Set

The following is the XML representation of the property set shown in Figure 43:

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

```
<PropertySet>
   <PropertySet
      AccountId="1-6"
      Name="Sara Chen"
      Phone="(415)298-7890"
      Location="San Francisco"
      AccessId="128" />
   <PropertySet
      AccountId="1-6"
      Name="Eric Brown"
      Phone="(650)123-1000"
      Location="Palo Alto"
      AccessId="129" />
</PropertySet>
```

Update

The Update method is called when a record is modified. Figure 44 illustrates the property set for Update input.

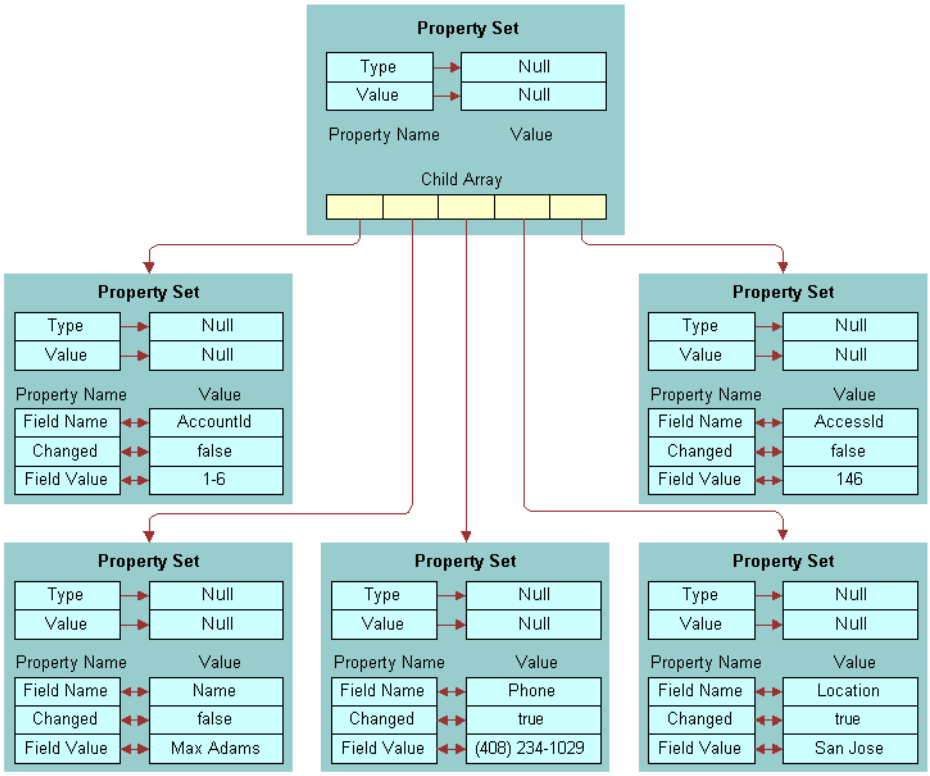


Figure 44. Update Input Property Set

The following is the XML representation of the property set shown in Figure 44:

```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet
    Business_spcComponent_spcId="1"
    Business_spcComponent_spcName="Contact">
    <PropertySet
        Field_spcName="AccountId"
        Changed="false"
        Field_spcValue="1-6"/>
    <PropertySet
```

```
        Field_spcName="Name"
        Changed="false"
        Field_spcValue="MaxAdams"/>
    <PropertySet
        Field_spcName="Phone"
        Changed="true"
        Field_spcValue="(408)234-1029"/>
    <PropertySet
        Field_spcName="Location"
        Changed="true"
        Field_spcValue="SanJose"/>
    <PropertySet
        Field_spcName="AccessId"
        Changed="false"
        Field_spcValue="146" />
</PropertySet>
```

Figure 45 illustrates the property set for the Update output.

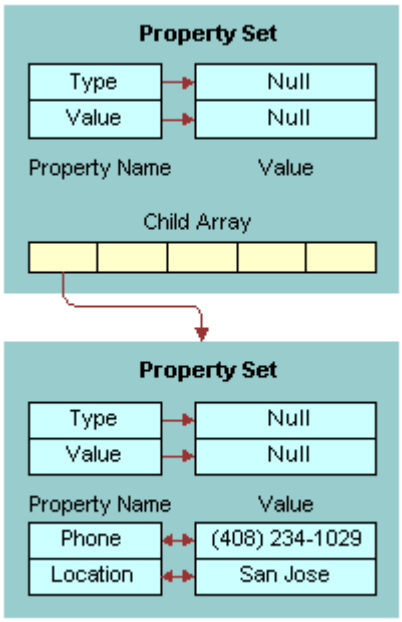


Figure 45. Update Output Property Set

The following is the XML representation of the property set shown in Figure 45:

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

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

<PropertySet

    <PropertySet

        Phone=="(408)234-1029"

        Location="San Jose" />

</PropertySet>
```

Custom Business Service Example

The following is an example of the 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:

```
function Service_PreInvokeMethod (MethodName, Inputs, Outputs)
{
 if (MethodName == "Init") {
     return(Init(Inputs, Outputs));
```

```
 }
 else if (MethodName == "Query") {
     return(Query(Inputs, Outputs));
 }
 else if (MethodName == "PreInsert") {
     return(PreInsert(Inputs, Outputs));
 }
 else if (MethodName == "Insert") {
     return(Insert(Inputs, Outputs));
 }
 else if (MethodName == "Update") {
     return(Update(Inputs, Outputs));
 }
 else if (MethodName == "Delete") {
     return(Delete(Inputs, Outputs));
 }
 else {
     return (ContinueOperation);
 }
}

function Init (Inputs, Outputs)
{
 // For debugging purpose...
 logPropSet(Inputs, "InitInputs.xml");
 Outputs.SetProperty("AccountId", "");
 Outputs.SetProperty("Name", "");
 Outputs.SetProperty("Phone", "");
 Outputs.SetProperty("AccessId", "");
 Outputs.SetProperty("Location", "");
 // For debugging purpose...
 logPropSet(Outputs, "InitOutputs.xml");
 return (CancelOperation);
}

function Query(Inputs, Outputs)
{
 // For debugging purpose...
 logPropSet(Inputs, "QueryInputs.xml");
 var selectStmt = "select * from Contacts ";
 var whereClause = "";
  var orderbyClause = "";
 // You have the following properties if you want to use them
 // Inputs.GetProperty("Business Component Name")
 // Inputs.GetProperty("Business Component Id")
 // Inputs.GetProperty("Remote Source")
 // If you configured Maximum Cursor Size at the buscomp,
 // get max-rows property
 var maxRows = Inputs.GetProperty("max-rows");
 // get search-string
 var searchString = Inputs.GetProperty("search-string");
  if (searchString != "" )
  {
 // convert the search-string into a where clause
 searchString = stringReplace(searchString, '*', '%');
 searchString = stringReplace(searchString, '[', ' ');
```

```
searchString = stringReplace(searchString, ']', ' ');
searchString = stringReplace(searchString, '~', ' ');
searchString = stringReplace(searchString, '"', "'");
 whereClause = " where ";
 whereClause = whereClause + searchString;
}
// match, search-spec, sort-spec
var childCount = Inputs.GetChildCount();
var child, sortProp;
for (var i = 0; i < childCount; i++)
{
    child = Inputs.GetChild(i);
    if (child.GetType() == "")
    {
    // Use this child property set if you want to use the old match field list.
    // We are not using this in this example.  We'll use search-string instead.
    }
    else if (child.GetType() == "search-spec")
    {
    // Use this child property set if you want to use the hierarchical
    // representation of the search-string.
    // We are not using this in this example.  We'll use search-string instead.
    }
    else if (child.GetType() == "sort-spec")
    {
    // This child property set has the sort spec.  We'll use this in this example
           orderbyClause = " order by ";
  var sortFieldCount = child.GetChildCount();
    for (var j = 0; j < sortFieldCount; j++)
    {
        // compose the order by clause
        sortProp = child.GetChild(j);
        orderbyClause += sortProp.GetProperty("field");
        var sortOrder = sortProp.GetValue();
        if (sortOrder == "DESCENDING")
            orderbyClause += " desc";
        if (j < sortFieldCount-1)
            orderbyClause += ", ";
        }
    }
}
// Now, our complete select statement is...
selectStmt += whereClause + orderbyClause;
// Now, query the data source
var conn = getConnection();
var rs = getRecordset();
rs.Open(selectStmt, conn);
// We're only going to return no more than maxRows of records.
var count = rs.RecordCount();
if (maxRows != "")
    if (count > maxRows)
        count = maxRows
// We'll go through the recordset and add them to the Outputs PropertySet.
var fcount, fields, row;
```

```
 for (i = 0; i < count; i++)
 {
     row = TheApplication().NewPropertySet();
     fields = rs.Fields();
     fcount = fields.Count();
     for (j = 0; j < fcount; j++)
     {
     var fieldValue = fields.Item(j).Value();
     if (fieldValue == null)
         row.SetProperty(fields.Item(j).Name(), "");
     else
         row.SetProperty(fields.Item(j).Name(), fieldValue);
     }
     Outputs.AddChild(row);
     rs.MoveNext();
 }
 // For debugging purpose...
 logPropSet(Outputs, "QueryOutputs.xml" );
 // clean up
 child = null;
 sortProp = null;
 row = null;
 rs.Close();
 rs = null;
 conn.Close();
 conn = null;
 return (CancelOperation);
}

function PreInsert (Inputs, Outputs)
{
 // For debugging purpose...
 logPropSet(Inputs, "PreInsertInputs.xml");
 var defaults = TheApplication().NewPropertySet();
 defaults.SetProperty("Location", "KO");
 Outputs.AddChild(defaults);
 // For debugging purpose...
 logPropSet(Outputs, "PreInsertOutputs.xml");
 // clean up
 defaults = null;
 return (CancelOperation);
}

function Insert (Inputs, Outputs)
{
 // For debugging purpose...
 logPropSet(Inputs, "InsertInputs.xml");
 var fieldList = "";
 var valueList = "";
 // Inputs should have only 1 child property set.
 var child = Inputs.GetChild(0);
 var fieldName = child.GetFirstProperty();
 var fieldValue;
 while (fieldName != "")
 {
     fieldValue = child.GetProperty(fieldName);
```

```
    if (fieldValue != "")
    {
    if (fieldList != "")
    {
        fieldList += ", ";
        valueList += ", ";
    }
    fieldList += fieldName;
    valueList += "'" + fieldValue + "'";
    }
    fieldName = child.GetNextProperty();
}
// The insert statement is...
var insertStmt = "insert into Contacts (" + fieldList + ") values (" + valueList + ")";
// Now, inserting into the data source...
var conn = getConnection();
conn.Execute (insertStmt);
// In this example, we need to query back the record just inserted to get
// the value of its primary key.  We made this primary key part of the buscomp
// to make update and delete easy.  The primary key is "AccessId".
var selectStmt = "select * from Contacts where ";
var whereClause = "";
child = Inputs.GetChild(0)
fieldName = child.GetFirstProperty();
while (fieldName != "")
{
    fieldValue = child.GetProperty(fieldName);
    if (fieldName != "AccessId")
    {
    if (whereClause != "")
        whereClause += " and ";
    if (fieldValue == "")
        whereClause += fieldName + " is null";
    else
        whereClause += fieldName + "='" + fieldValue + "'";
    }
    fieldName = child.GetNextProperty();
}
// The select statement is...
selectStmt += whereClause;
// Now, let's select the new record back
var rs = getRecordset();
rs.Open(selectStmt, conn);
// We're expecting only one row back in this example.
var fcount, fields, row, fieldValue;
row = TheApplication().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");
 // clean up
 child = null;
 row = null;
 rs.Close();
 rs = null;
 conn.Close();
 conn = null;
 return (CancelOperation);
}

function Update (Inputs, Outputs)
{
 // For debugging purpose...
 logPropSet(Inputs, "UpdateInputs.xml");
 var child;
 var childCount = Inputs.GetChildCount();
 var fieldName, fieldValue;
 var updateStmt = "update Contacts set ";
 var setClause = "";
 var whereClause;
 // Go through each child in Inputs and construct
 // necessary sql statements for update and query
 for (var i = 0; i < childCount; i++)
 {
     child = Inputs.GetChild(i);
     fieldName = child.GetProperty("Field Name");
     fieldValue = child.GetProperty("Field Value");
     // We only need to update changed fields.
     if (child.GetProperty("Changed") == "true")
     {
         if (setClause != "")
             setClause += ", ";
         if (fieldValue == "")
             setClause += fieldName + "=null";
         else
             setClause += fieldName + "='" + fieldValue + "'";
     }
     if (fieldName == "AccessId")
         whereClause = " where AccessId = " + fieldValue;
 }
 // The update statement is...
 updateStmt += setClause + whereClause;
 // Now, updating the data source...
 var conn = getConnection();
 conn.Execute (updateStmt);
 // How to construct the Outputs PropertySet can vary, but in this example
 // We'll query back the updated record from the data source.
 var selectStmt = "select * from Contacts" + whereClause;
 // Now, let's select the updated record back
 var rs = getRecordset();
```

```
 rs.Open(selectStmt, conn);
 // We're expecting only one row back in this example.
 // In this example, we're returning all the fields and not just
 // the updated fields.  You can only return those updated
 // fields with the new value in the Outputs property set.
 var fcount, fields, row, fieldValue;
 row = TheApplication().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, "UpdateOutputs.xml");
 // clean up
 child = null;
 row = null;
 rs.Close();
 rs = null;
 conn.Close();
 conn = null;
 return (CancelOperation);
}

function Delete (Inputs, Outputs)
{
 // For debugging purpose...
 logPropSet(Inputs, "DeleteInputs.xml");
 // Inputs should have only 1 child property set.
 var child = Inputs.GetChild(0);
 // In this example, we're only using the AccessId
 // (it's the primary key in the Contacts db)
 // for delete statement for simplicity.
 var deleteStmt = "delete from Contacts where AccessId = " + child.GetProperty("AccessId");
 // Now, let's delete the record from the data source.
 var conn = getConnection();
 conn.Execute(deleteStmt);
 // For debugging purpose...
 logPropSet(Outputs, "DeleteOutputs.xml");
 // Returning empty Outputs property set.
 // clean up
 conn.Close();
 conn = null;
 return (CancelOperation);
}
```

The following functions are helper functions:

```
function getConnection ()
{
```

```
 // VBCContact is the ODBC data source name
 var connectionString = "DSN=VBCContact";
 var uid = "";
 var passwd = "";
 var conn = COMCreateObject("ADODB.Connection");
 conn.Mode = 3;
 conn.CursorLocation = 3;
 conn.Open(connectionString , uid, passwd);
 return conn;
}

function getRecordset()
{
 var rs = COMCreateObject("ADODB.Recordset");
 return rs;
}

function logPropSet(inputPS, fileName)
{
 // Use EAI XML Write to File business service to write
 // inputPS property set to fileName file in c:\temp directory.
 var fileSvc = TheApplication().GetService("EAI XML Write to File");
 var outPS = TheApplication().NewPropertySet();
 var fileLoc = "c:\\temp\\" + fileName;
 var tmpProp = inputPS.Copy();
 tmpProp.SetProperty("FileName", fileLoc);
 fileSvc.InvokeMethod("WritePropSet", tmpProp, outPS);
 // clean up
 outPS  = null;
 fileSvc = null;
 tmpProp = null;
}

function stringReplace (string, from, to)
{
 // Replaces from with to in string
 var stringLength = string.length;
 var fromLength = from.length;
 if ((stringLength == 0) || (fromLength == 0))
 return string;
 var fromIndex = string.indexOf(from);
 if (fromIndex < 0)
     return string;
 var newString = string.substring(0, fromIndex) + to;
 if ((fromIndex + fromLength) < stringLength)
     newString += stringReplace(string.substring(fromIndex+fromLength, stringLength), from, to);
 return newString;
}
```

For more examples of VBCs, see *Developing and Deploying Siebel Business Applications*.

9 Siebel EAI and File Attachments

Siebel EAI supports file attachments for exchanging business documents such as sales literature, activity attachments, and product defect attachments with another Siebel instance or an external system such as Oracle Applications.

The chapter includes the following topics:

About File Attachments

For example, if you are exchanging service requests with another application or partner, you can include attachments such as screen captures, email, log files, and contract agreements that are associated with the service request in the information being exchanged. Siebel EAI support for file attachments allows comprehensive integration.

In order to use file attachments you first need to create Integration Objects. For 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.

Exchange of Attachments with External Applications

Siebel EAI supports bidirectional attachments exchange with external applications using the following two message types:

- **MIME (Multipurpose Internet Mail Extensions).** MIME is the industry standard for exchanging multipart messages. The first part of the MIME message is an XML document representing the business object being exchanged and attachments to the object are included as separate parts of the multipart message. MIME is the recommended choice for integrating Siebel Business Applications with other applications.

- **Inline XML attachments (Inline Extensible Markup Language).** With inline XML attachments, the entire business object you are exchanging, including any attachments, is sent as a single XML file. In this case, attachments are included within the body of the inline XML attachment. Consider using inline XML attachments when integrating two instances of Siebel Business Applications using file attachments. For information, see *XML Reference: Siebel Enterprise Application Integration*.

Using MIME Messages to Exchange Attachments

To send or receive file attachments using MIME messages, Siebel EAI uses the MIME Hierarchy Converter and MIME Doc Converter.

The following checklist shows the high-level procedures you must perform to use MIME to exchange attachments between Siebel Business Applications and another external system:

- Create an integration object using the EAI Siebel Wizard.

  For information, see "Creating an Attachment Integration Object" on page 210.

- Create an inbound or outbound Workflow process.

  For information, see "Creating Workflow Process Examples" on page 212.

- Test your workflow process using Workflow Process Simulator.

  For information, see "About the EAI MIME Hierarchy Converter" on page 216.

Creating an Attachment Integration Object

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

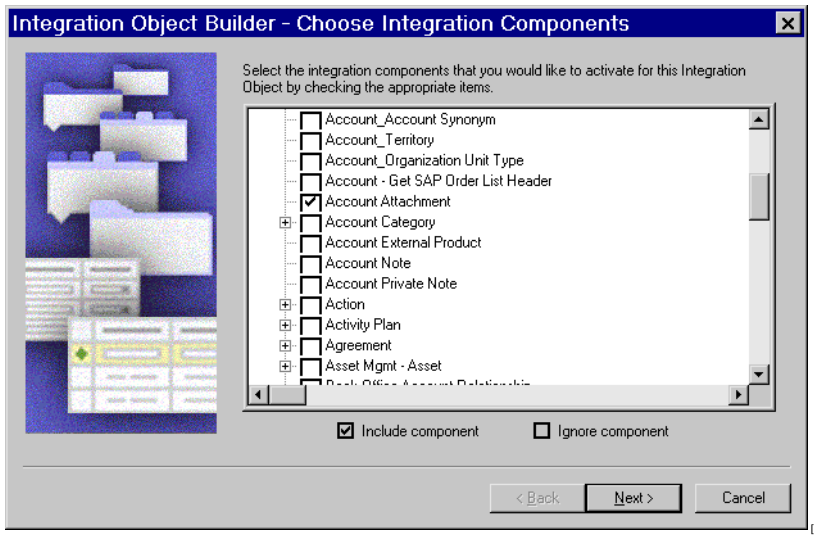
To create a new attachment integration object

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. Choose File > 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" on page 38 to create the new integration object, for example *SourceObject* Attachment.

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



5 In the Object Explorer, select Integration Object, and then select your new integration object in the Object List Editor.

6 In the Object Explorer, expand the Integration Object tree to show the Integration Component object.

7 Select the *SourceObject* Attachment integration component, and set its External Sequence and XML Sequence properties so that they are greater than those of the other integration components (that is, last in sequence), if not already set.

If they are not last, the situation can arise where the attachment is processed successfully (and the file system is physically updated). Then a subsequent integration component causes a failure (for example, an attempt to insert to the database causes a duplicate error). In this case, the database transaction is rolled back, but the file system is not restored.

8 With the *SourceObject* Attachment integration component selected, expand the Integration Component object, and then select the Integration Component Field object.

The Integration Components and Integration Component Fields lists appear.

9 Inactivate all integration component fields except the following:

- *SourceObject* Attachment Id, for example, Accnt Attachment Id
- *SourceObject*FileExt, for example, AccntFileExt
- *SourceObject*FileName, for example, AccntFileName
- Comment

10 Select the *SourceObject* Attachment Id component field, and then verify that its Data Type property is set to DTYPE_ATTACHMENT.

11 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 Process Examples

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

Outbound Workflow Process

To process the attachment for an outbound request you must create a workflow process to query the database, convert the Integration Object and its attachments into a MIME hierarchy, and then create a MIME document to send to the File Transport business service.

To create an outbound workflow process

1 Navigate to the Siebel Business Process Designer.

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

3 Define your process properties.

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

Name	Data Type	Default String
SiebelMessage	Hierarchy	Leave blank.
Error Message	String	Leave blank.
Error Code	String	Leave blank.
Object Id	String	Leave blank.
Process Instance Id	String	Leave blank.
Siebel Operation Object Id	String	Leave blank.
MIMEHierarchy	Hierarchy	Leave blank.
SearchSpec	String	[Account.Name] = 'Sample Account'
<Value>	String	Default output is binary.

4 The first business service queries the Account information from the database using the EAI Siebel Adapter business service with the Query method. This step requires the following input and output arguments.

Input Argument	Type	Value	Property Name	Property Data Type
Output Integration Object Name	Literal	Sample Account	n/a	n/a
SearchSpec	Process Property	n/a	SearchSpec	String

Property Name	Type	Output Argument
SiebelMessage	Output Argument	Siebel Message

NOTE: For more information on using the EAI Siebel Adapter, see Chapter 6, "EAI Siebel Adapter."

5 The second business service in the workflow converts the Account integration object and its attachments to a MIME hierarchy using the EAI MIME Hierarchy Converter business service with the SiebelMessage to MIME Hierarchy method. This step requires the following input and output arguments.

Input Argument	Type	Property Name	Property Data Type
Siebel Message	Process Property	SiebelMessage	Hierarchy

Property Name	Type	Output Argument
MIMEHierarchy	Output Argument	MIME Hierarchy

NOTE: For more information on the EAI MIME Hierarchy Converter, see "About the EAI MIME Hierarchy Converter" on page 216.

6 The third business service of the workflow converts the MIME hierarchy to a document to be sent to File Transport business service. This step uses the EAI MIME Doc Converter business service with the MIME Hierarchy To MIME Doc method. This step requires the following input and output arguments.

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

Property Name	Type	Output Argument
<Value>	Output Argument	MIME Message

NOTE: For more information on the EAI MIME Doc Converter, see "About the EAI MIME Doc Converter" on page 218.

7 For the final step, set up the last business service of the workflow to write the information into a file using the EAI File Transport business service with the Send method. This step requires the following input arguments.

Input Argument	Type	Value	Property Name	Property Data Type
Message Text	Process Property	n/a	<Value>	String
File Name	Literal	c:\temp\account.txt	n/a	n/a

NOTE: For information on File Transport, see *Transports and Interfaces: Siebel Enterprise Application Integration*.

Inbound Workflow Process Example

To process the attachment for an inbound request, you must create a workflow process 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 process from an external application (or another business service) as the input property set, the corresponding property name in the input property set must be same name as the process property and is case sensitive.

To create an inbound workflow process

1 Navigate to Workflow Process.

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

3 Define your process properties.

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

Name	Data Type
SiebelMessage	Hierarchy
Error Message	String
Error Code	String
Object Id	String
Siebel Operation Object Id	String
MIMEHierarchy	Hierarchy
MIMEMsg	String

4 The first business service in the workflow reads the Account information from a file using the EAI File Transport business service with Receive method. This step requires the following input and output arguments.

Input Argument	Type	Value
File Name	Literal	c:\temp\account.txt

Property Name	Type	Output Argument
<Value>	Output Argument	Message Text

NOTE: For information on File Transport, see *Transports and Interfaces: Siebel Enterprise Application Integration*.

5 The second business service of the workflow converts the Account information to a MIME hierarchy using the EAI MIME Doc Converter business service with the MIME Doc to MIME Hierarchy method. This step requires the following input and output arguments.

Input Argument	Type	Property Name	Property Data Type
MIME Message	Process Property	<Value>	String

Property Name	Type	Output Argument
MIMEHierarchy	Output Argument	MIME Hierarchy

6 The third business service of the workflow converts the MIME hierarchy to a document, and sends it to the EAI Siebel Adapter business service. This step uses the EAI MIME Hierarchy Converter business service with the MIME Hierarchy to Siebel Message method. This step requires the following input and output arguments.

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

Property Name	Type	Output Argument
SiebelMessage	Output Argument	Siebel Message

7 The last step of the workflow writes the information into the database using the EAI Siebel Adapter business service with the Insert or Update method. This step requires the following input argument.

Input Argument	Type	Property Name	Property Data Type
Siebel Message	Process Property	SiebelMessage	Hierarchy

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 46 illustrates the Siebel Message of a sample Account with attachments. This figure represents both input and output to the MIME Hierarchy Converter.

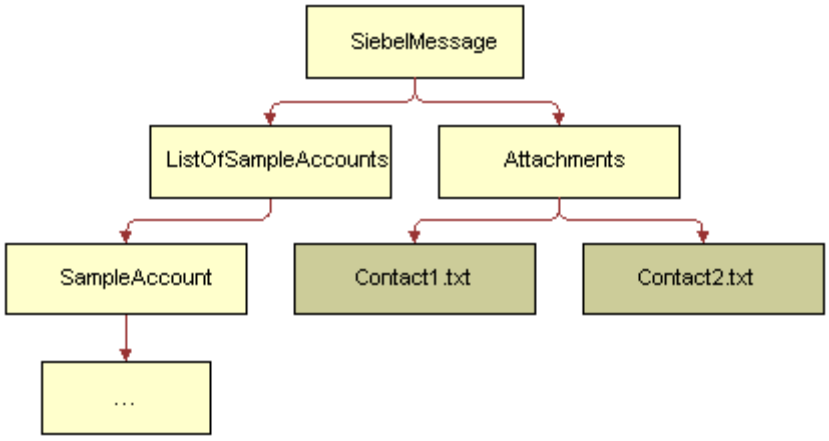


Figure 46. Sample Account with Attachments as Input to the MIME Hierarchy Converter

The output of this process is illustrated in Figure 47.

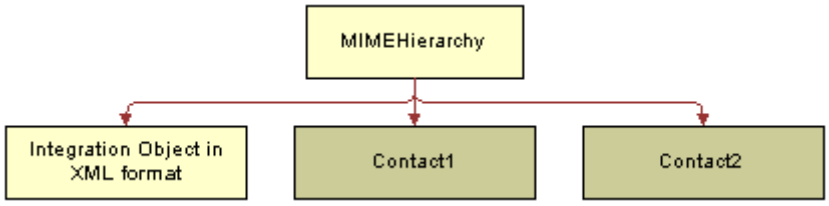


Figure 47. Output of a MIME Hierarchy Converter

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, 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 48 illustrates the incoming hierarchy.

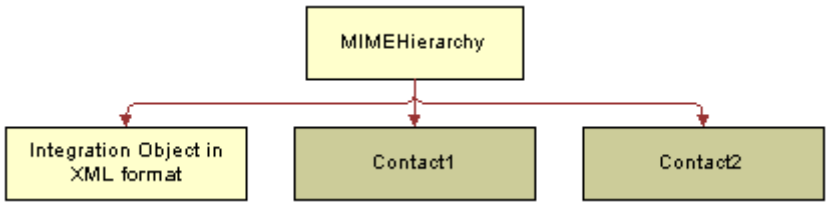


Figure 48. Output of a MIME Hierarchy Converter

The output of this process is illustrated in Figure 46 on page 217. The output for this process is the same as the input.

About the EAI MIME Doc Converter

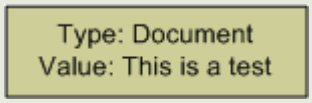
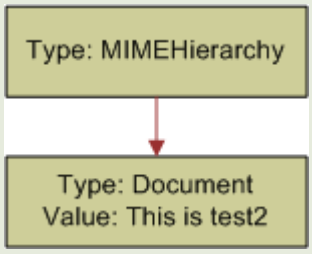
The MIME Doc Converter converts a MIME Hierarchy into a MIME Message and a MIME Message into a MIME Hierarchy. A MIME Hierarchy consists of two different types of property sets.

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

EAI MIME Doc Converter Properties

Table 45 illustrates some examples of how a MIME Message maps to a MIME Hierarchy.

Table 45. Examples of MIME Message and MIME Hierarchy

MIME Message	MIME Hierarchy
MIME-Version: 1.0 Content-Type: application/xml Content-Transfer-Encoding: 7bit This is a test.	
MIME-Version: 1.0 Content-Type: multipart/related; type="application/xml"; boundary=--abc ----abc Content-Type: application/xml Content-Transfer-Encoding: 7bit This is test2. ----abc--	

The business service needs the following properties on the child property set as shown in Table 46. These properties reflect the most accurate information on the data contained in the child property set.

Table 46. Properties for EAI MIME Doc Converter

Property	Possible Values	Type	Description
ContentId	Any value	Document	No Default. The ContentId is the value used to identify the file attachment when the receiver parses the MIME message. When importing attachments, use a unique value for this property and not repeat it for the rest of the file attachments. This is required in the actual document. This property is automatically populated when you are exporting an attachment from a Siebel application.
Extension	txt, java, c, C, cc, CC, h, hxx, bat, rc, ini, cmd, awk, html, sh, ksh, pl, DIC, EXC, LOG, SCP, WT, mk, htm, xml, pdf, AIF, AIFC, AIFF, AU, SND, WAV. gif, jpg, jpeg, tif, XBM, avi, mpeg, ps, EPS, tar, zip, js, doc, nsc, ARC, ARJ, B64, BHX, GZ, HQX	Document	No Default. If ContentType and ContentSubType are not defined, the Extension is used to retrieve the appropriate values from this property. If all three values are specified, the ContentType and ContentSubType values override the values retrieved from the Extension. If either the Extension or both ContentType and ContentSubType are not specified, the ContentType will be set to application and ContentSubType will have the value of octet-stream.

Table 46. Properties for EAI MIME Doc Converter

Property	Possible Values	Type	Description
ContentType	application, audio, image, text, video	Document	Default is application. The ContentType value has to be specified if you want to set the content type of the document instead of using the extension to get a value from the MIME utility function. If the value is not provided, the default value is used. The ContentType of multipart is used to represent file attachments in a MIME message. Other forms of values to describe a multipart is not supported.
ContentSubType	plain, richtext, html, xml (used with ContentType of Text) octet-stream, pdf, postscript, x-tar, zip, x-javascript, msword, x-conference, x-gzip (used with ContentType of application) aiff, basic, wav (used with ContentType of audio) gif, jpeg, tiff, x-xbitmap (used with ContentType of image) avi, mpeg (used with ContentType of video)	Document	Default is octet-stream. The ContentSubType value has to be specified if you want to set the content subtype of the document instead of using the extension to get a value from the MIME utility function. If the value is not provided the default value is used.

NOTE: On the inbound direction, the business service is independent of the transport. It assumes that the input property set contains the MIME message, and writes a property set representation of the MIME message. A property set is used to represent each part of the MIME message. When decoding the MIME message, the business service automatically sets the properties based on the values in the MIME message.

Using Inline XML to Exchange Attachments

To exchange attachments between applications, you use the EAI Siebel Adapter business service:

- To send a message to an external application, call the EAI Siebel Adapter with an integration object that has an integration component from an attachment business component. The EAI Siebel Adapter generates the integration object hierarchy and then converts it to an XML document. The attachment is included in the XML in the *SourceObject*FileBuffer element.
- To insert an attachment into a Siebel 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 process:

- Creating an attachment integration object using the EAI Siebel Wizard business service

  For information, see "Creating an Attachment Integration Object" on page 210.

  CAUTION: To avoid SQL errors, you must inactivate all integration component fields in the integration object except those in Step 9 on page 211.

- "Creating an Attachment" on page 222
- "Creating a Test Workflow Process" on page 223
- Testing your workflow process using the Workflow Process 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 Process

You create a workflow process in Siebel Tools to do the following:

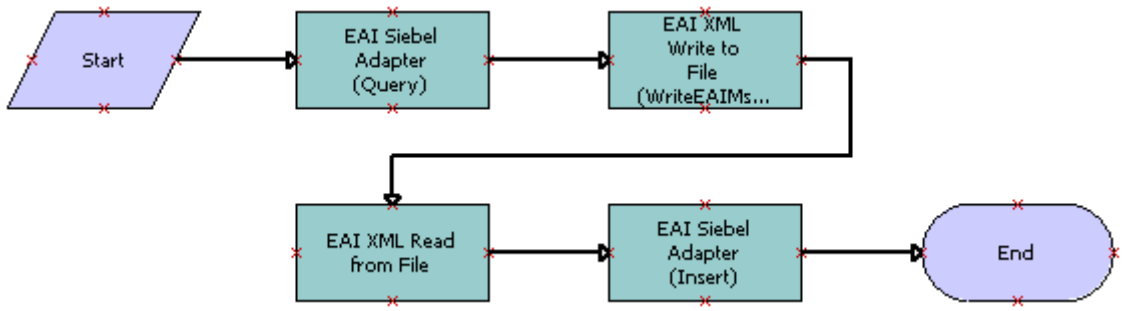
- Query the Siebel database for the record with the attachment.
- Convert the integration object and its attachment into a Siebel Message.
- Read an external XML file (containing an attachment) and convert it into a Siebel Message.

  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 process to exchange attachments

1 Create the following workflow process:



2 Define the process properties:

Name	Data Type
Error Code	String
Error Message	String
Object Id	String
Process Instance Id	String
Siebel Operation Object Id	String
SiebelMessage	Hierarchy

3 The first business service step queries the database using the EAI Siebel Adapter business service with the Query method. This step requires the following input and output arguments:

Input Argument	Type	Value
OutputIntObjectName	Literal	Attachment integration object you created, for example, Account Attachment
PrimaryRowId	Literal	Row ID of the record to which you added an attachment

Property Name	Type	Output Argument
SiebelMessage	Output Argument	SiebelMessage

4 The second business service step writes the integration object hierarchy to an XML file using the EAI XML Write to File business service with the WriteEAIMsg method. This step requires the following input arguments:

Input Argument	Type	Value
FileName	Literal	File to write, for example, d:\temp\AttachmentTest_write.xml
SiebelMessage	Process Property	SiebelMessage

5 The third business service step reads an XML hierarchy and converts it into a Siebel Message using the EAI XML Read From File business service with the ReadEAIMsg method. This step requires the following input and output arguments:

Input Argument	Type	Value
FileName	Literal	File to read, for example, d:\temp\AttachmentTest_read.xml **NOTE:** For testing purposes, you can use a modified form of the file written in the second business step, which will automatically have the correct hierarchy.

Property Name	Type	Output Argument
SiebelMessage	Output Argument	SiebelMessage

6 The fourth business service step reads the Siebel Message and inserts the record into the Siebel database using the EAI Siebel Adapter business service with the Insert method. This step requires the following input argument:

Input Argument	Type	Value
SiebelMessage	Process Property	SiebelMessage

10 External Business Components

The external business component feature provides a way to access data that resides in a non-Siebel table or view, using a Siebel business component.

This chapter consists of the following topics:

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 233. To configure EBCs, you perform the following high-level tasks:

- "Creating the External Table Definition" on page 227.

  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.

  As may be appropriate, 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.

- "Mapping External Columns to Siebel System Fields" on page 231.

  Map columns in the external table or view to Siebel 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.

- "Specifying the Data Source Object" on page 232.

  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 may need to be exposed in the Object Explorer. See the *Object Types Reference* on the *Siebel Bookshelf* for information on exposing objects in the Object Explorer of Siebel Tools.

  This object tells the object manager which data source to use to access the object.

- "Specifying Any Optional Table Properties" on page 233.

  When the table is imported, specify additional table properties for the corresponding external table.

- "Configuring the External Business Component" on page 233.

  Configure the EBC and specify the data source object. This data source name will be the same as that specified for the Table object.

- "Specifying Run-Time Parameters" on page 234.

  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.

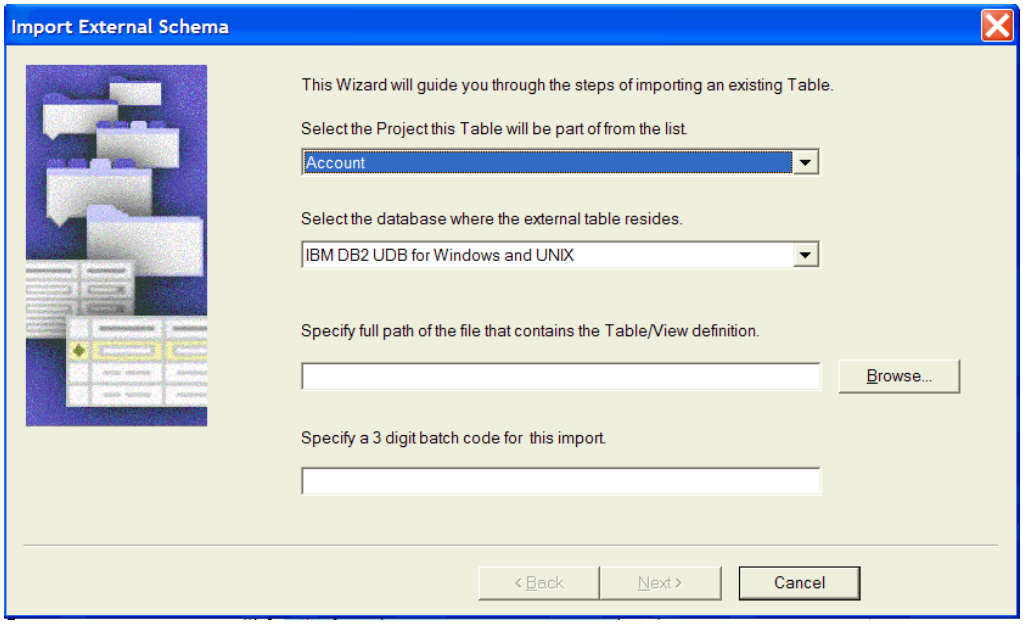
For more information about using Siebel Tools, see *Using Siebel Tools* on the *Siebel Bookshelf*.

This task is a step in "Process of Configuring External Business Components" on page 226.

To import the external table definition

1. Start Siebel Tools.
2. In Siebel Tools, check out and lock the appropriate project.
3. Select File > New Object....
4. In the New Object Wizards applet, on the General tab, double-click External Table Schema Import.

   The External Table Schema Import Wizard appears, as shown in the following figure.



5. In the External Table Schema Import Wizard, specify the following values:
   - The project the new Table object definition will be associated with.
   - The database where the external table resides. The value specified must correspond to the database platform used by the Siebel database.
   - The full path for the location of the SQL/DDL file that contains the external table definition.
   - Specify the three-digit batch code that allows grouping.

6 Click Next to confirm the entries, and 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 3 through Step 6 for every external table definition you want to import.

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 47 contains the data type mappings you can use when importing table definitions.

Table 47. Supported Data Type Mappings by Product

Supported Data Types	Siebel Data Type
MS SQL Server Data Types	
int	Numeric with scale of 0
bigint	Numeric with scale of 0
smallint	Numeric with scale of 0
tinyint	Numeric with scale of 0
float	Numeric
real	Numeric
decimal	Numeric
money	Numeric
smallmoney	Numeric
bit	Character with a length of 1
char	Character
nchar	Character
varchar	Varchar
nvarchar	Varchar
text	Long
ntext	Long
datetime	Date Time
smalldatetime	Date Time
DB2 Universal Database Data Types	
UINT	Numeric with scale of 0
BIGUINT	Numeric with scale of 0
SMALLUINT	Numeric with scale of 0

Table 47. Supported Data Type Mappings by Product

Supported Data Types	Siebel Data Type
FLOAT	Numeric
REAL	Numeric
DECIMAL	Numeric
NUMERIC	Numeric
CHAR	Character
VARGRAPHIC	Varchar
LONG VARGRAPHIC	Long
DATE	Datetime
TIME	Datetime
TIMESTAMP	Datetime
Oracle Data Types	
Number	Numeric
TIMESTAMP WITH TIME ZONE	Numeric
TIMESTAMP WITH LOCAL TIME ZONE	Numeric
Char	Character
Nchar	Character
varchar2	Varchar
nvarchar2	Varchar
Long	Long
date	Datetime
Oracle Business Intelligence (BI) Server Data Types	
Integer	Numeric with scale of 0
Smallint	Numeric with scale of 0
Tinyint	Numeric with scale of 0
Float	Numeric
Double	Numeric
Bit	Character (1)
Boolean	Character (1)
Char	Character

Table 47. Supported Data Type Mappings by Product

Supported Data Types	Siebel Data Type
Varchar	Varchar
Longvarchar	Long
Datetime	Datetime
Date	Datetime
Time	Datetime

Table 48 contains the data types that are not supported for importing table definitions.

Table 48. Unsupported Data Type Mappings by Product

Database	Unsupported Data Types
MS SQL Server	timestamp
	varbinary
	binary
	image
	cursor
	uniqueidentifier
DB2 Universal Database	CLOB
	DBCLOB
	BLOB
Oracle	TIMESTAMP
	CLOB
	NCLOB
	BLOB
	BFILE
	ROWID
	UROWID
	RAW
	LONG RAW
	INTERVAL YEAR TO MONTH
	INTERVAL DAY TO SECOND

Table 48. Unsupported Data Type Mappings by Product

Database	Unsupported Data Types
Oracle BI Server	Timestamp
	Varbinary
	Longvarbinary
	Binary
	Object
	Unknown

About the New Imported Table Definition

After the table definition is imported using the External Table Schema Import Wizard, the external table and the external column names are generated.

The external table name is stored in the Table object's Alias property. This external table name consists of the following:

- An EX prefix (for external table)
- A three-digit batch code specified in the External Table Schema Import Wizard
- An automatically generated seven-digit number

An example of the Table name is:

```
EX_ABC_0000001
```

The external column name is stored in the Column child object's Alias property. An X is added as the prefix and a four-digit number is added as the suffix for the external column name, for example:

```
X_ABC_0000001_0001
```

The Table object's Type property is set to External or External View (if a view was imported). This column denotes that the table resides outside of the Siebel database.

Mapping External Columns to Siebel System Fields

This task is a step in "Process of Configuring External Business Components" on page 226.

When the EBC is defined, you must map the Siebel application's system fields to the corresponding external table column. System Field mapping is accomplished at the column definition, whether 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 application's 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 internal fields.

The following is a list of the Siebel application's internal fields that may 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 should 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 226.

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, 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

This task is a step in "Process of Configuring External Business Components" on page 226.

When the table is imported, you may 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 should be handled by an external API. If this property is set to TRUE, the BusComp_PreWriteRecord and BusComp_PreDeleteRecord events should be scripted 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, 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 invoked when generating a primary key for a business component.

For more information about these table properties, see *Siebel Tools Online Help*.

Configuring the External Business Component

This task is a step in "Process of Configuring External Business Components" on page 226.

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:

- The Data Source business component property is specified when configuring an EBC. The value specified for this property should correspond with the name of the corresponding Table Data Source.
- The Log Changes property is set to false (unchecked, the default is true). This will prevent Siebel Remote or Replication transactions from being created.
- When configuring a many-to-many relationship, the intersection table resides in the same database instance as the child table.
- It is recommended that all EBCs use the CSSBusComp class.

NOTE: Substituting a Siebel-provided table with an external table may result in significant downstream configuration work, and in some cases may restrict or prevent the use of standard functionality provided for the Siebel Business Applications.

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 testing using the Siebel Developers Web Client, add a [DataSource] section to the client .cfg file.

This task is a step in "Process of Configuring External Business Components" on page 226.

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 Administration - Server Configuration > Enterprises > 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 should specify the TNS name associated with the database and not an ODBC or other entry.
   - DSSQLStyle = *database SQL type*

     See Table 49 on page 236 for a listing of the supported SQL types.
   - DSDLLName = *DLL Name corresponding to the SQL type*

     See Table 49 on page 236 for a listing of the supported connector DLL names and SQL Styles.
   - DSTableOwner = *data source table owner*
   - DSUsername = *default user name 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 log in prompt when accessing the external data source, specify *DSUsername* and the *DSPassword*. If specified, this will override the default user name and password.

The DSUsername and the DSPassword parameters are activated only when using the Database Security Adapter. For more information, see "Configuring a User in LDAP or ADSI Security Adapter To Access EBCs."

Configuring a User in LDAP or ADSI Security Adapter To Access EBCs

It is not a good idea to assume that the login to your primary data source is always the same as the login to the external data source.

If you are using the Lightweight Directory Access Protocol (LDAP) or the Microsoft Active Directory Service Interfaces (ADSI) setup for the Siebel application and you try to access an EBC, the security adapter is called to authorize the user trying to access the external database.

When LDAP or ADSI authentication is used, the username and password values for the external data source is provided in the ADSI `SharedCredentialsDN` parameter and the `CredentialAttributeType` attribute.

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 = mail
```

In your ADSI server modify the mail attribute for the following entry:

```
cn=sharedcred,ou=people,dc=siebel,dc=com
```

Before modifying, one value must already exist (assuming *`sadmin`* and *`db2`* are the user name and password for the *`ServerDataSrc`* data source, which is the primary data source):

```
type=ServerDataSrc username=sadmin password=db2
```

Add additional values to the mail 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 mail attribute, you are able to access EBCs.

Configuring the Data Source Definition for the Siebel Developers Web Client

If testing using the Siebel Developers Web Client, 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 49 on page 236):

```
[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 DLL Names and SQL Styles

When defining the DLL and SQL files for importing the external schema, the external database being used might not be the same as the Siebel database. Table 49 contains the supported connector DLL names and the corresponding SQL styles.

Table 49. Supported Connector DLL Names and SQL Styles

External Databases	DLL Names	SQL Styles
IBM DB2	sscddcli.dll	DB2
Microsoft SQL Server	sscdms80.dll	MSSqlServer
Oracle. Use for Oracle databases.	sscdo90.dll	Oracle
Oracle. Use for Oracle databases with cost-based optimization.	sscdo90.dll	OracleCBO **NOTE:** Use OracleCBO only with an Oracle 9i (or later) instance. The parameter values used are different in earlier versions.
Oracle Business Intelligence (BI) Server	sscdsacon.dll	Siebel Analytics Server
SQL Anywhere	sscdw8.dll	Watcom

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 Administration - Server Configuration > Enterprises > 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. Select 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 datasource specified in the "To configure the data source definition" section.

   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.

About 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)

About the IsNewRecordPending Business Component Method

This method can be invoked by using a script in the PreWriteRecord event to determine if the current record is newly created. If the record is a new record, this method returns the value TRUE.

An example script for the use of this method follows:

```
var    isNewRecord = this.InvokeMethod("IsNewRecordPending");
```

About the GetOldFieldValue Business Component Method

This method can be invoked by using a script in the PreWriteRecord event to retrieve an old field value if needed. This invoke method takes an input parameter, which must be a valid field name, and returns a string containing the old field value.

An example script for the use of this method follows:

```
var    oldLoc = this.InvokeMethod("GetOldFieldValue", "Location");
```

About the 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 invoke method takes an input parameter of TRUE or FALSE.

An example script for the use of this method follows:

```
var    requery = this.InvokeMethod("SetRequeryOnWriteFlag", "TRUE");
```

About the 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 invoke method, SetRequeryOnWriteFlag, can be invoked again with the FALSE parameter to reset the requery on write mode, if needed.

An example script for the use of this method follows:

```
var    extWrite = this.InvokeMethod("SetRequeryOnWriteFlag", "TRUE");
// insert script here to commit the record via an mechanism channel
var    resetWrite = this.InvokeMethod("SetRequeryOnWriteFlag", "FALSE");
```

Usage and Restrictions for External Business Components

The following usage guidelines and restrictions apply to EBCs:

- Creating and populating the external table is the responsibility of the customer. Consult your database administrator for the appropriate method to use.
- EBCs cannot be docked, so they do not apply to mobile users on the Siebel Mobile Web Client. Siebel Remote is not supported.
- EBCs support many-to-many relationships with the limitation that for such relationships the intersection table must be from the same data source as the child business component.
- EBCs cannot be loaded using the Enterprise Integration Manager.

- EBCs rely on the Business Object Layer of the Siebel Architecture. Therefore, EBCs are used only in Siebel Server components using this layer such as the Application Object Manager (for example, the Call Center Object Manager), Workflow Process Manager, and so on. EBCs are not used on components not using this layer such as Workflow Policies (the Workflow Monitor agent), Assignment Manager, Incentive Compensation, and so on.
- The Id field must be mapped to an underlying column in the external table in order 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 data base.
- If the column that was mapped to the Id system field has *Primary Key* checked, then rowid values are generated by the object manager. Otherwise, a user-entered rowid value is assumed, and the object manager does not generate a rowid value for it.

  However, in either configuration, the Primary Key column should not use the Oracle Sequence Object.
- For EBCs that contain multi-value groups, if a primary join is enabled, then both the parent and the child business components must be from the same data source. Multi-value 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.

NOTE: Significant configuration effort and changes may 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 may 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, new data sources corresponding to the data sources specified for the external tables need to 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, a log-in window appears to initiate logging into the external data source.

About Overriding Connection Pooling Parameters for the DataSource

Overriding the connection pooling parameters for the DataSource is supported. If connection pooling is enabled for the component, but should be turned off for the data source, 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 datasource

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 may be specified. However, increasing the number of external joined data sources can cause degradation in performance.

About 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.

Performance tests are recommended if searching and sorting are permitted on the fields based on joins to the external tables. The Siebel application does not have information of 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 may produce competing results. In that case, Step 2 takes precedence.

If, as result of the search and sort specifications in effect, 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 should be queried are supported. These directives may be applied based on a knowledge of the data shape in the Siebel and external tables.

For example, using the following property values:

Property	Value
External DataSource Field Priority On Search: FieldA	1
External DataSource Field Priority On Search: FieldB	2

A query on Field A is likely to be selective. If there is a search specification on Field A, 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, 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 Using 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.

Loading an Oracle Business Intelligence Presentation Catalog for Use as an External Table

An EBC is a tool that derives its data from an external relational data source. In Siebel Tools, the structure of the external table is imported using the External Table Schema Import wizard.

EBCs, in conjunction with the Oracle Business Intelligence (BI) database connector, allow the ability to construct business components that derive their data from Oracle BI Enterprie Edition. EBCs support Oracle BI as a source for having Siebel Tools import the structure of an Oracle BI Presentation Catalog by reading sources such as an XML file.

NOTE: Oracle BI integration is read-only; any business components that use Oracle BI as a data source must be configured to support read-only access.

The following procedure generates the Oracle BI Presentation Catalog as an XML file from Oracle BI. Follow these instructions to avoid the Repository Documentation wizard exporting the full repository definition into an XML file, and not only the selected object.

To load an Oracle BI Presentation Catalog for use as an external table

1. Start the Oracle Business Intelligence Administration Tool.
2. Choose File > New and create a new repository file.
3. Choose File > Import from Repository:
   a. Choose the appropriate repository and click Next.
   b. When requested, type the username and password.
   c. Select Catalog from the drop-down list.
   d. Choose the catalog you want to import into Siebel Tools, then click Add With Children.
   e. Click Next.
   f. Click Finish.
4. Choose Tools > Utilities, choose Repository Documentation, and then click Execute….
5. In the Save as Type field, select XML as the file extension.
6. Give a new name to the file, then click Save.

If you import an XML file that contains several Presentation Catalogs, Siebel Tools creates one external table for each catalog.

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 should 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 Administration >Server Configuration > Servers > Components > Events and 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.

A Predefined EAI Business Services

Siebel Business Applications provide a number of business services. These services do not require any modification, but they do require that you choose and configure them to suit your requirements.

For general information on using business services, see Chapter 4, "Business Services."

Table 50 presents the predefined Siebel EAI business services.

Table 50. Predefined EAI Business Services

Business Service	Class	Description
EAI XSD Wizard	CSSXMLSchemaWizard	Used to create integration objects based on XSD files.
EAI XML XSD Generator	CSSEAISchXSDService	Used to generate an XSD file from an integration object.
EAI Transaction Service	CSSBeginEndTransactionService	EAI Transaction service for working with Siebel transactions, such as begin and end, to find out whether in transaction.
EAI MSMQ Transport	CSSMsmqTransService	EAI MSMQ Transport.
EAI MQSeries Server Transport	CSSMqSrvTransService	EAI MQSeries Server Transport.
EAI HTTP Transport	CSSHTTPTransService	EAI HTTP Outbound Transport. For information, see *Transports and Interfaces: Siebel Enterprise Application Integration*.
EAI Siebel Adapter	CSSEAISiebelAdapterService	EAI Siebel Adapter. For information, see Chapter 6, "EAI Siebel Adapter."
EAI Query Spec Service	CSSEAIQuerySpecService	Used internally by the EAI Siebel Adapter to convert the SearchSpec method argument as a string to an Integration Object Instance that the EAI Siebel Adapter can use as a Query By Example object.
EAI Import Export	CSSEAIImportExportService	EAI Import Export Service (import and export integration object from or to XML).
EAI BTS COM Transport	CSSEAIBtsComService	EAI Siebel to BTS COM Transport.

Table 50. Predefined EAI Business Services

Business Service	Class	Description
EAI DLL Transport	CSSDllTransService	EAI DLL Transport. For information, see *Transports and Interfaces: Siebel Enterprise Application Integration*.
EAI Data Transformation Engine	CSSDataTransformationEngine	EAI Data Transformation Engine. For information, see *Business Processes and Rules: Siebel Enterprise Application Integration*. The display name for this business service is EAI Data Mapping Engine.
EAI Null Envelope Service	CSSEAINullEnvelopeService	EAI Null Envelope Service. For information, see *XML Reference: Siebel Enterprise Application Integration*.
Siebel Message Envelope	CSSEAISMEnvelopeService	EAI Siebel Message Envelope Service. For information, see *XML Reference: Siebel Enterprise Application Integration*.
EAI Dispatch Service	CSSEAIDispatchService	Dispatch Service. For information, see *Business Processes and Rules: Siebel Enterprise Application Integration*.
EAI Integration Object to XML Hierarchy Converter	CSSEAIIntObjHierCnvService	EAI Integration Object Hierarchy (also known as SiebelMessage) to XML hierarchy converter service. For information, see *XML Reference: Siebel Enterprise Application Integration*.
EAI MIME Hierarchy Converter	CSSEAIMimePropSetService	EAI MIME Hierarchy Conversion Service. For information, see Chapter 9, “Siebel EAI and File Attachments.”
EAI MIME Doc Converter	CSSEAIMimeService	MIME Document Conversion Service. For information, see Chapter 9, “Siebel EAI and File Attachments.”
EAI XML Converter	CSSEAIXMLCnvService	Converts between XML and EAI Messages. For information, see *XML Reference: Siebel Enterprise Application Integration*.
EAI XML Write to File	CSSEAIXMLPrtService	Print a property set to a file as XML. For information, see *XML Reference: Siebel Enterprise Application Integration*.

Table 50. Predefined EAI Business Services

Business Service	Class	Description
EAI XML Read from File	CSSEAIXMLPrtService	Read an XML file and parse to a property set. For information, see *XML Reference: Siebel Enterprise Application Integration*.
XML Converter	CSSXMLCnvService	Converts between XML documents and arbitrary Property Sets. For information, see *XML Reference: Siebel Enterprise Application Integration*.
XML Hierarchy Converter	CSSXMLCnvService	Converts between XML documents and XML Property Set or Arbitrary Property Set. For information, see *XML Reference: Siebel Enterprise Application Integration*.

B Property Set Representation of Integration Objects

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

This appendix consists of the following topics:

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 51.

Table 51. Property Set Node Types

Name	Parent	Value of Type Attribute	Properties	Description
Service Method Arguments	Not applicable	Ignored	The properties of this property set contain the service specific parameters, such as *PrimaryRowId* for the EAI Siebel Adapter.	This is the top-level property set of a business service's input or output. The properties of this property set contain the service-specific parameters (for example, *PrimaryRowId* for the EAI Siebel Adapter).
SiebelMessage	Service Method Arguments	SiebelMessage	The properties of this property set contain header attributes associated with the integration object, for example, *IntObjectName*.	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*.
Object List	SiebelMessage	List*OfObjectType*	Not used.	This property set identifies the object type that is being represented. The root components of the object instances are children of this property set.

Table 51. Property Set Node Types

Name	Parent	Value of Type Attribute	Properties	Description
Root Component	Object List	*Root Component Name*	The property names of the property set represent the field names of the component, and the property values are the field values.	This property set represents the root component of an integration object instance.
Child Component Type	Root Component or Component	ListOf*Component Name*	Not used.	An integration component can have a number of child component types, each of which can have zero or more instances. The Integration Object Hierarchy format groups the child components of a given type under a single property set. This means that child components are actually grandchildren of their parent component's property set.
Child Components	Child Component Type	*Component Name*	The property names of the property set represent the field names of the component, and the property values are the field values.	This property set represents a component instance. It is a grandchild of the parent component's property set.

Example Instance of an Account Integration Object

This example shows an Account integration object in which the object has two component types: Account and Business Address (which is a child of Account). The hierarchy of component types from the perspective of Oracle's Siebel Tools, looks like that shown in Figure 49.

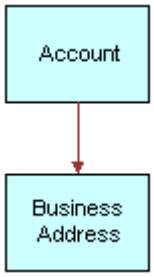


Figure 49. Sample Account Integration Object

Figure 50 on page 253 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.

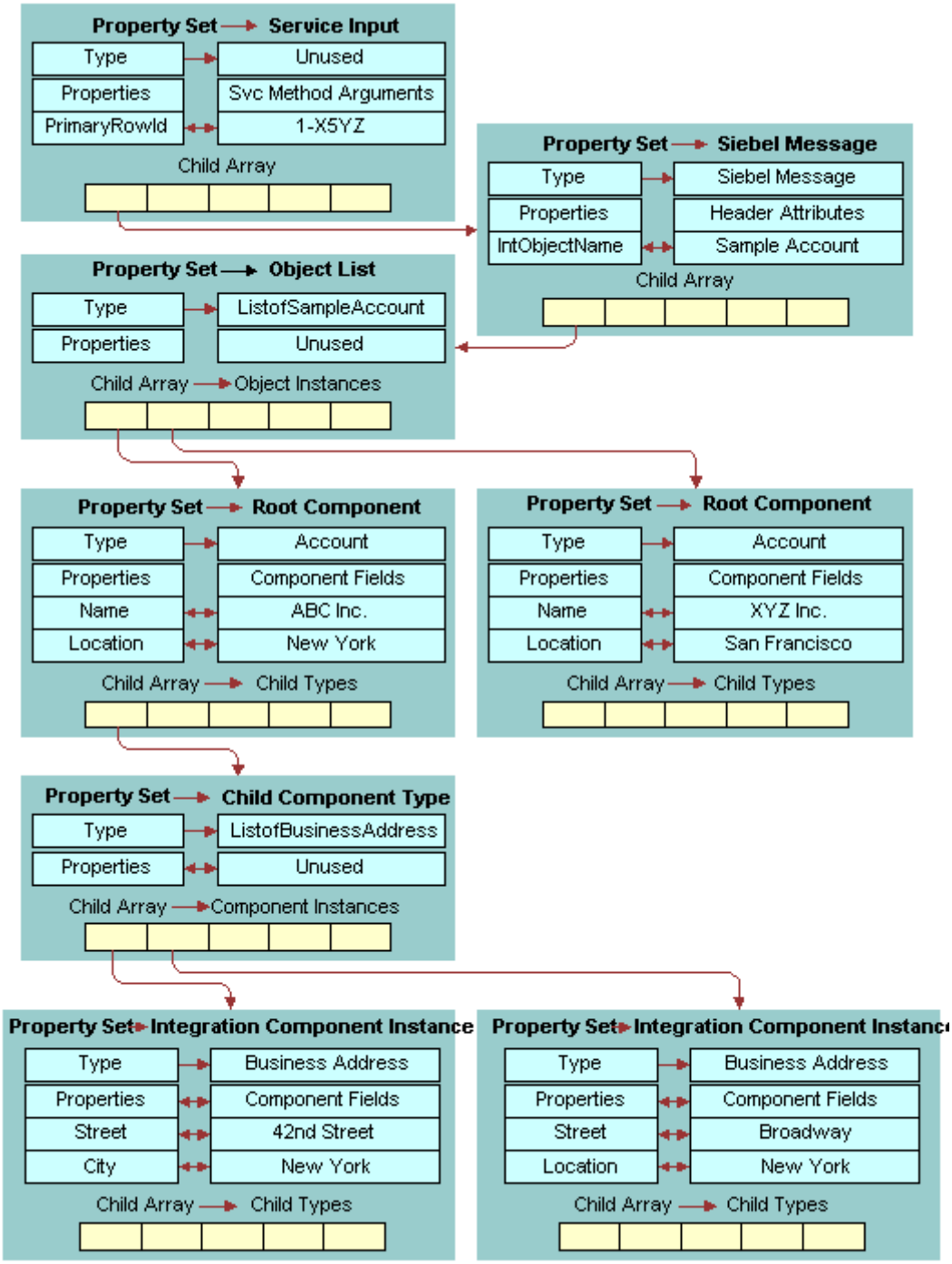


Figure 50. Partial Instance of Sample Account Integration Object

C DTDs for XML Gateway Business Service

This appendix lists the various inbound and outbound DTDs for the XML Gateway business service. It covers the following topics:

Outbound DTDs for the XML Gateway Business Service

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

Delete

The following DTD is for the Delete request:

```
<!ELEMENT siebel-xmlext-delete-req  (buscomp, remote-source, row)>

<!ELEMENT buscomp (#PCDATA)>

<!ATTLIST buscomp id NMTOKEN #REQUIRED>

<!ELEMENT remote-source ( #PCDATA )*>

<!ELEMENT row (value+)>

<!ELEMENT value (#PCDATA)*>

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

Init

The following DTD is for the Init request:

```
<!ELEMENT siebel-xmlext-fields-req (buscomp, remote-source?)>

<!ELEMENT buscomp (#PCDATA)>

<!ATTLIST buscomp id NMTOKEN #REQUIRED >

<!ELEMENT remote-source (#PCDATA)*>
```

Insert

The following DTD is for the Insert request:

```
<!ELEMENT siebel-xmlext-insert-req (buscomp, remote-source?, row)>

<!ELEMENT buscomp (#PCDATA)>

<!ATTLIST buscomp id NMTOKEN #REQUIRED>

<!ELEMENT remote-source (#PCDATA)*>

<!ELEMENT row (value+)>

<!ELEMENT value (#PCDATA)*>

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

PreInsert

The following DTD is for the PreInsert request:

```
<!ELEMENT siebel-xmlext-preinsert-req (buscomp, remote-source?)>

<!ELEMENT buscomp (#PCDATA)>

<!ATTLIST buscomp id NMTOKEN #REQUIRED >

<!ELEMENT remote-source (#PCDATA)*>
```

Query

The following DTD is for the Query request:

```
<!ELEMENT siebel-xmlext-query-req (buscomp , remote-source?, max-rows?, search-
string?, match?, search-spec?, sort-spec? )>

<!ELEMENT buscomp (#PCDATA)>

<!ATTLIST buscomp id NMTOKEN #REQUIRED>

<!ELEMENT remote-source (#PCDATA)*>

<!ELEMENT max-rows (#PCDATA)>

<!ELEMENT search-string (#PCDATA)>

<!ELEMENT match (#PCDATA)>

<!ATTLIST match field CDATA #REQUIRED>

<!ELEMENT search-spec (node)>

<!ELEMENT node (#PCDATA | node)*>

<!ATTLIST node node-type (Constant | Identifier | Unary Operator | Binary Operator)
#REQUIRED>

<!ATTLIST node value-type (TEXT | NUMBER | DATETIME | UTCDATETIME | DATE | TIME)
#IMPLIED>
```

```
<!ELEMENT sort-spec (sort+)>

<!ELEMENT sort (#PCDATA)>

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

Update

The following DTD is for the Update request:

```
<!ELEMENT siebel-xmlext-update-req  (buscomp, remote-source?, row)>

<!ELEMENT buscomp (#PCDATA)>

<!ATTLIST buscomp id NMTOKEN #REQUIRED>

<!ELEMENT remote-source (#PCDATA)*>

<!ELEMENT row (value+)>

<!ELEMENT value (#PCDATA)*>

<!ATTLIST value changed ( true | false ) #REQUIRED>

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

Inbound DTDs for the XML Gateway Business Service

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

Delete Response

The following DTD is for the Delete response:

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

Init Response

The following DTD is for the Init response:

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

<!ELEMENT support EMPTY >

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

Insert Response

The following DTD is for the Insert response:

```
<!ELEMENT siebel-xmlext-preinsert-ret (row)>
<!ELEMENT row (value+)>
<!ELEMENT value (#PCDATA)*>
<!ATTLIST value field CDATA  #REQUIRED >
```

PreInsert Response

The following DTD is for the PreInsert response:

```
<!ELEMENT siebel-xmlext-preinsert-ret (row)>
<!ELEMENT row (value)*>
<!ELEMENT value (#PCDATA)*>
<!ATTLIST value field CDATA  #REQUIRED >
```

Query Response

The following DTD is for the Query response:

```
<!ELEMENT siebel-xmlext-query-ret (row*)>
<!ELEMENT row (value+)>
<!ELEMENT value (#PCDATA)*>
<!ATTLIST value field CDATA #REQUIRED >
```

Update Response

The following DTD is for the Update response:

```
<!ELEMENT siebel-xmlext-update-ret (row)>
<!ELEMENT row (value+)>
<!ELEMENT value (#PCDATA)>
<!ATTLIST value field CDATA  #REQUIRED >
```

Index

F

H

I

J

L

M

P

Q

R

S

T

U

V

W

X