# Integration Platform Technologies: Siebel eBusiness Application Integration Volume II

Version 7.7, Rev. A

September 2004

# Contents

# 1 What’s New in This Release

## What’s New in Integration Platform Technologies: Siebel eBusiness Application Integration Volume II, Version 7.7, Rev. A

Table 1 lists changes described in this version of the documentation to support Release 7.7 of the software.

Table 1. New Product Features in Integration Platform Technologies: Siebel eBusiness Application Integration Volume II, Version 7.7, Rev. A

| Topic | Description |
|---|---|
| “Picklist Validation” on page 58 | Information is added between using the Object Manager or the Siebel EAI Adapter to validate picklists. |
| “Supported Connector DLL Names and SQL Styles” on page 181 | Table added listing the supported connector DLL names and the corresponding SQL styles to use when configuring external business components. |

## What’s New in Integration Platform Technologies: Siebel eBusiness Application Integration Volume II, Version 7.7

Table 2 lists changes described in this version of the documentation to support Release 7.7 of the software.

Table 2. New Product Features in Integration Platform Technologies: Siebel eBusiness Application Integration Volume II, Version 7.7

| Topic | Description |
|---|---|
| “Permission Flags for Integration Components” on page 35<br><br>“About the EAI Siebel Adapter Access Control” on page 35 | Sections moved from Chapter 3, “Creating and Maintaining Integration Objects” to Chapter 2, “Integration Objects.” |
| “About RPC-Literal and DOC-Literal Bindings” on page 74 | The Web Services infrastructure has been enhanced in release 7.7 to provide one-way operations, RPC-literal support and DOC-literal support, and custom SOAP filters. |
| “About Local Business Service” on page 85 | Web Services utilize specialized SOAP headers for common tasks such as authentication, authorization, and logging. A Local Business Service, as a transport option for outbound Web Services, is supported in the Siebel application. |

Table 2. New Product Features in Integration Platform Technologies: Siebel eBusiness Application Integration Volume II, Version 7.7

| Topic | Description |
|---|---|
| “About Web Services Security Support” on page 93 | Release 7.7 introduces support for the UserName Token mechanism of the WS-Security specification, which allows Siebel applications to send and receive user credentials in a standards-compliant manner. |
| “About Web Services Cache Refresh” on page 98 | Administrators are able to update the definitions of both Siebel Inbound and Outbound Web Services to provide more current or correct functionality by using the Web Services Cache Refresh. |
| “Enabling Web Services Tracing” on page 98 | Enhanced section to include all of the event types that can be used for Web Services tracing. |
| Chapter 9, “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. |

# 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 using different terminology from one system to another. Table 3 has been included to clarify the information in this chapter by providing a standard terminology for these concepts.

Table 3. Integration Objects Terminology

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

Table 3. 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 some Siebel business object. |
| Integration message | A bundle of data consisting of two major parts: header information that describes what should be done with or to the message itself, and instances of integration objects, that is, data in the structure of the integration object. |

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

**NOTE:** For more information, see *Business Processes and Rules: Siebel eBusiness Application Integration Volume IV*.

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

When planning your integration project, you should consider several issues:

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

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

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

## Integration Object Base Object Type

Each integration object created in Siebel Tools has to be based on one of the base object types presented in Table 4. This property is used by adapters to determine whether the object is a valid object for them to process.

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

Table 4. Integration Object Base Object Types

| Base Object Type | Description |
|---|---|
| None | For internal use only. |
| OLE DB | Used to expose Siebel business component as OLEDB rowset that can be used by OLEDB consumers such as Excel, Word, and so on. The OLE DB Provider only accepts integration objects of this type. |
| SAP BAPI Input | Used to represent the input structure of an SAP RFC or BAPI function call. For details, see *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 details, see *Siebel Connector for SAP R/3 Guide*. |
| SAP IDOC | Used with the IDOC Adapter and Receiver in version 6.x and 7.0. For details, see *Siebel Connector for SAP R/3 Guide*. |
| SAP IDOC Adapter | Used to represent an SAP IDOC structure. For details, see *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. EAI Siebel Adapter only accepts integration object of this type. |
| Table | Obsolete. |
| XML | Used to represent external XML Schema such as DTD or XSD. For details on DTD and XSD, see *XML Reference: Siebel eBusiness Application Integration Volume V*. |

## 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 thing.

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

An integration object instance is actual data organized in the format or structure of the integration object. Figure 1 illustrates a simple example of an integration object and an integration object instance, using partial data.

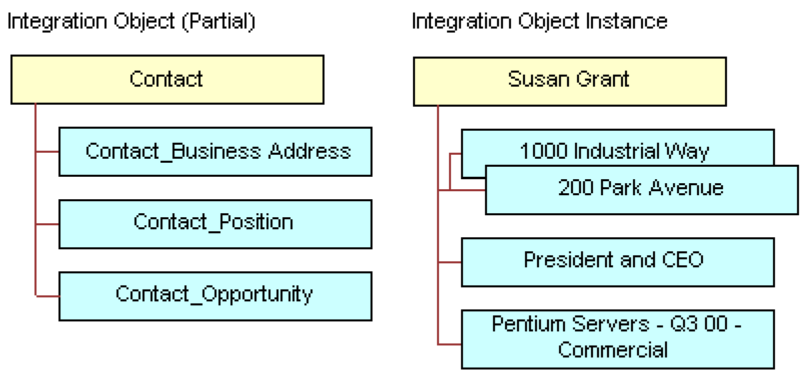


Figure 1. Integration Object and Integration Object Instance

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

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

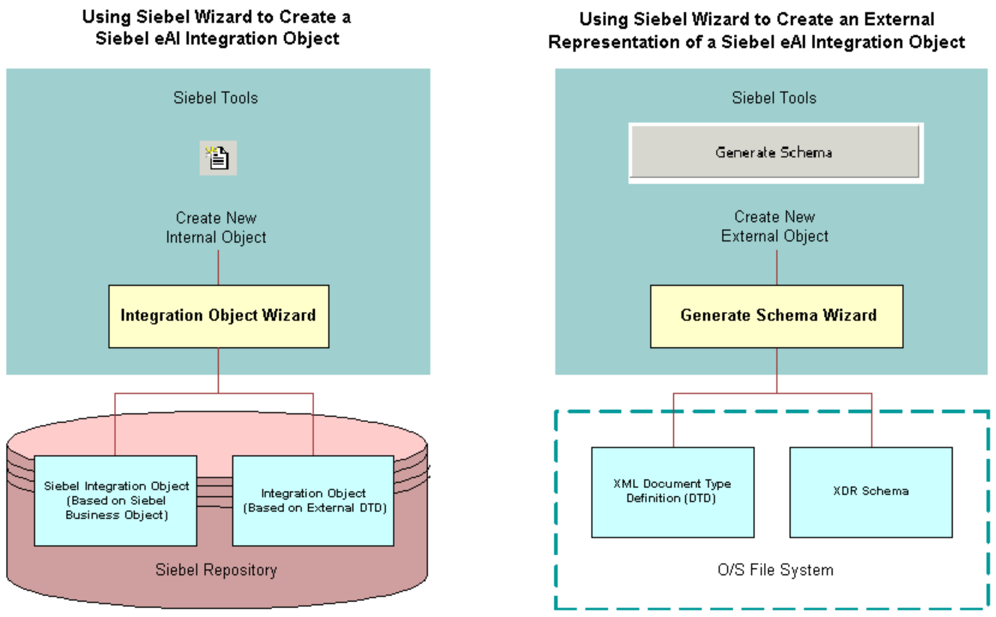

Using Siebel Wizard to Create a Siebel eAI Integration Object
Siebel Tools
Create New Internal Object
Integration Object Wizard
Siebel Integration Object (Based on Siebel Business Object)
Integration Object (Based on External DTD)
Siebel Repository
Using Siebel Wizard to Create an External Representation of a Siebel eAI Integration Object
Siebel Tools
Generate Schema
Create New External Object
Generate Schema Wizard
XML Document Type Definition (DTD)
XDR Schema
O/S File System

Figure 2. Integration Object Wizards

- **Integration Object Builder wizard.** This wizard lets you create a new object. It supplies the functionality for creating integration objects from Siebel business objects or integration objects based on representations of external business objects using XML Schema Definition (XSD) or Document Type Definition (DTD). To access this wizard, navigate to the New Object 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 eBusiness Application Integration documentation set where appropriate.

## About the Structure of Integration Objects

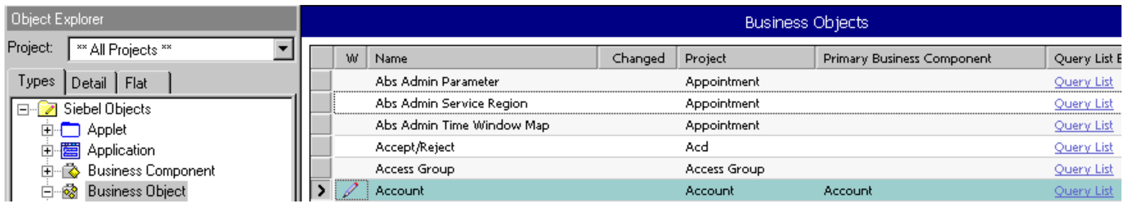
The Siebel integration object provides a hierarchical structure that represents a complex data type. Most specifically, prebuilt eAI integration objects describe the structure of Siebel business objects, SAP IDOCs, SAP BAPIs, XML, and external data. Most integration projects require the use of an integration 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" continues with descriptions of how to create integration objects. The initial process of using the Integration Object Builder wizard is essentially the same for every integration object type currently supported.

**CAUTION:** You should avoid using or modifying integration objects in the EAI Design project. Using or modifying any objects in the EAI Design project can cause unpredictable results.

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

An integration object consists of one Parent Integration Component, sometimes referred to as the root component or the primary integration component. The Parent Integration Component corresponds to the primary business component of the business object you chose as the model for your integration object. Figure 3 shows the Account business object in Siebel Tools.



| W | Name | Changed | Project | Primary Business Component | Query List E |
|---|---|---|---|---|---|
| | Abs Admin Parameter | | Appointment | | Query List |
| | Abs Admin Service Region | | Appointment | | Query List |
| | Abs Admin Time Window Map | | Appointment | | Query List |
| | Accept/Reject | | Acd | | Query List |
| | Access Group | | Access Group | | Query List |
| | Account | | Account | Account | Query List |

Figure 3. Account Parent Business Component

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

Each child component can have one or more child components. In Siebel Tools, if you look at the integration components for an integration object you have created, you will see that each component can have one or more fields. Figure 4 on page 18 illustrates a partial view of a Siebel integration object based on the Account business object, with the Business Address component and the Contact component activated.

Figure 4 represents part of the structure of the Account integration object. The Account parent integration component can have both fields and child integration components. Each integration component can also have child integration components and fields. A structure of this sort represents the *metadata* of an Account integration object. You may choose to inactivate components and fields. By inactivating components and fields, you can define the structure of the integration object instances entering or leaving the system.

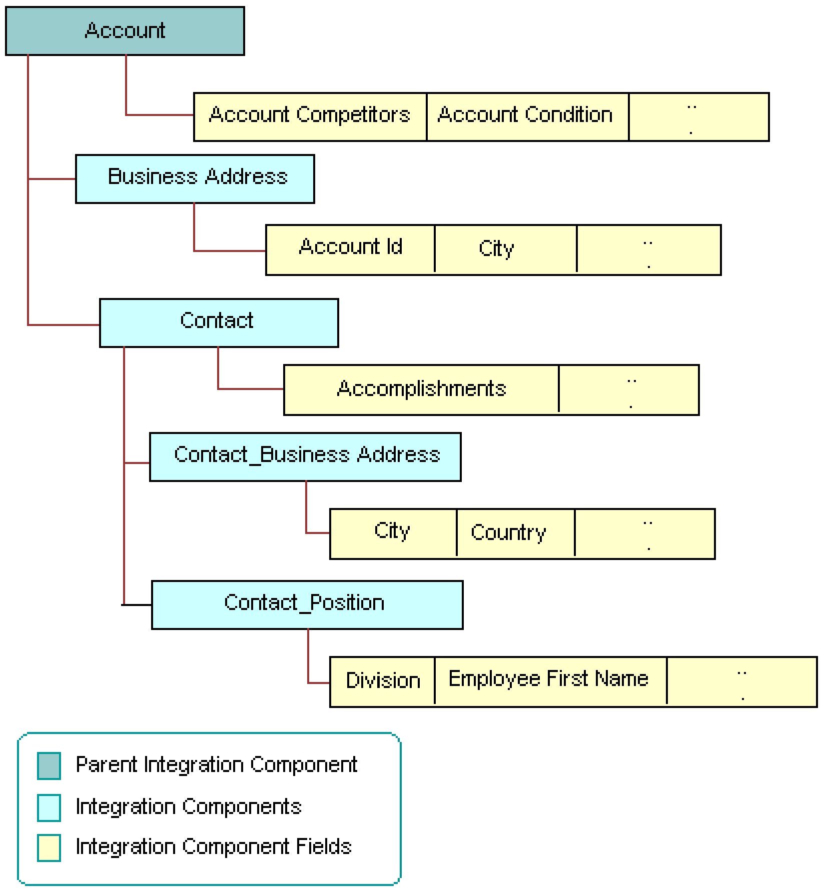
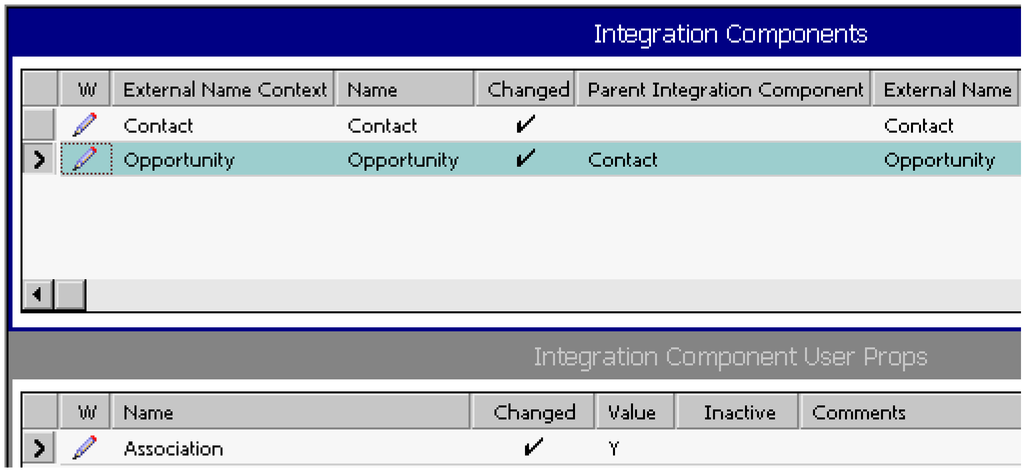


Figure 4. Representation of Partial Account Integration Object

## 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 5, the Opportunity integration component has one user property defined: the *Association* user property, set to a value of *Y*.



| W | External Name Context | Name | Changed | Parent Integration Component | External Name |
|---|---|---|---|---|---|
| | Contact | Contact | ✓ | | Contact |
| | Opportunity | Opportunity | ✓ | Contact | Opportunity |

| W | Name | Changed | Value | Inactive | Comments |
|---|---|---|---|---|---|
| | Association | ✓ | Y | | |

Figure 5. Integration Component Representation of Association

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

## About Multi-Value Groups Within Business Components

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

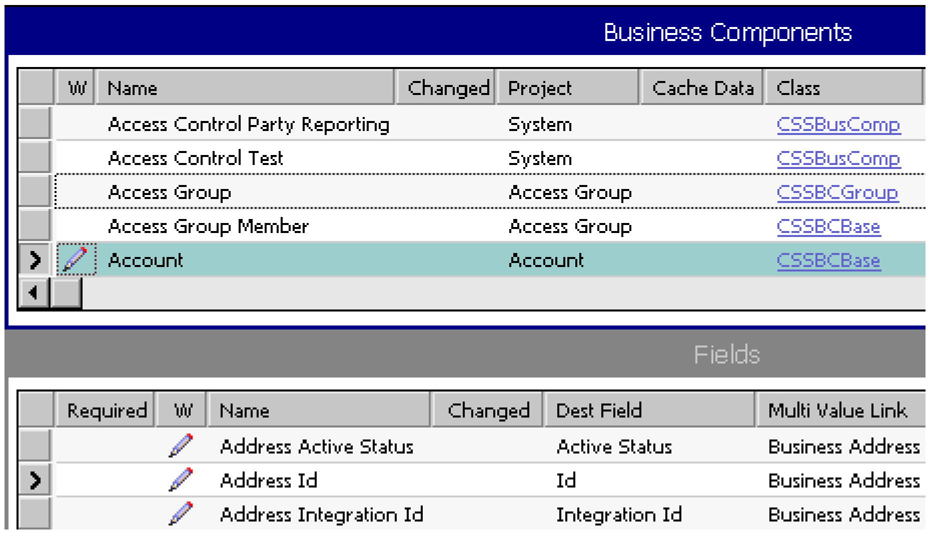
An integration object instance most often has multiple integration component instances. For example, an Account can have multiple Business Addresses but only one of these addresses is marked as the primary address. A business requirement may require that only the integration component instance that corresponds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

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

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

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

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


Business Components

| W | Name | Changed | Project | Cache Data | Class |
|---|---|---|---|---|---|
| | Access Control Party Reporting | | System | | CSSBusComp |
| | Access Control Test | | System | | CSSBusComp |
| | Access Group | | Access Group | | CSSBCGroup |
| | Access Group Member | | Access Group | | CSSBCBase |
| ✎ | Account | | Account | | CSSBCBase |

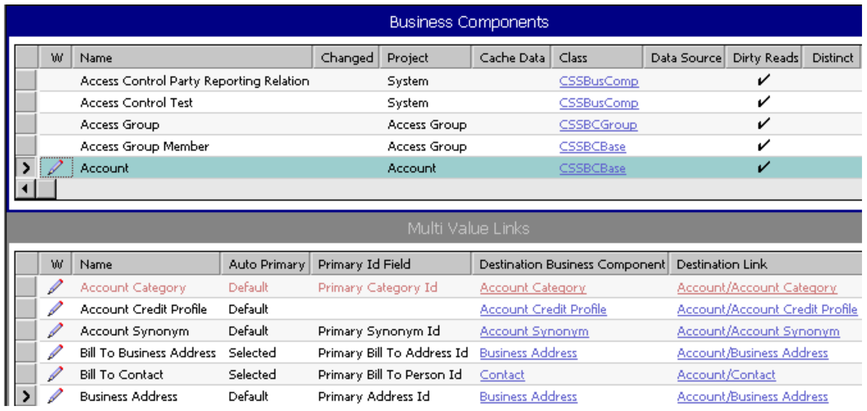
Fields

| Required | W | Name | Changed | Dest Field | Multi Value Link |
|---|---|---|---|---|---|
| | ✎ | Address Active Status | | Active Status | Business Address |
| | ✎ | Address Id | | Id | Business Address |
| | ✎ | Address Integration Id | | Integration Id | Business Address |

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

## Screen 2: Multi-Value Links in a Business Component

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



Business Components

| W | Name | Changed | Project | Cache Data | Class | Data Source | Dirty Reads | Distinct |
|---|---|---|---|---|---|---|---|---|
| | Access Control Party Reporting Relation | | System | | CSSBusComp | | ✓ | |
| | Access Control Test | | System | | CSSBusComp | | ✓ | |
| | Access Group | | Access Group | | CSSBCGroup | | ✓ | |
| | Access Group Member | | Access Group | | CSSBCBase | | ✓ | |
| ✎ | Account | | Account | | CSSBCBase | | ✓ | |

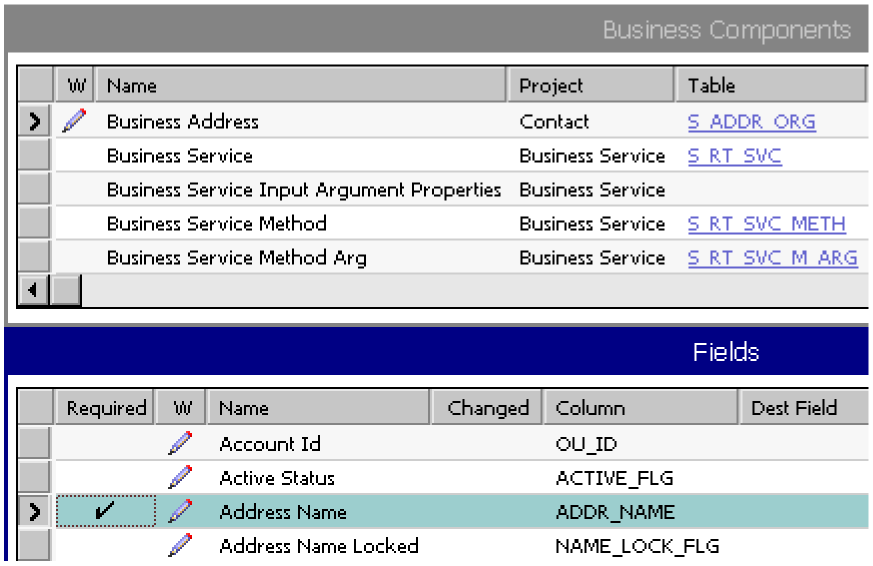
Multi Value Links

| W | Name | Auto Primary | Primary Id Field | Destination Business Component | Destination Link |
|---|---|---|---|---|---|
| ✎ | Account Category | Default | Primary Category Id | Account Category | Account/Account Category |
| ✎ | Account Credit Profile | Default | | Account Credit Profile | Account/Account Credit Profile |
| ✎ | Account Synonym | Default | Primary Synonym Id | Account Synonym | Account/Account Synonym |
| ✎ | Bill To Business Address | Selected | Primary Bill To Address Id | Business Address | Account/Business Address |
| ✎ | Bill To Contact | Selected | Primary Bill To Person Id | Contact | Account/Contact |
| ✎ | Business Address | Default | Primary Address Id | Business Address | Account/Business Address |

Figure 7. Destination Business Component

## Screen 3: Fields View After Adding Multi-Value Link

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



Business Components

| W | Name | Project | Table |
|---|---|---|---|
| ✎ | Business Address | Contact | S_ADDR_ORG |
| | Business Service | Business Service | S_RT_SVC |
| | Business Service Input Argument Properties | Business Service | |
| | Business Service Method | Business Service | S_RT_SVC_METH |
| | Business Service Method Arg | Business Service | S_RT_SVC_M_ARG |

Fields

| Required | W | Name | Changed | Column | Dest Field |
|---|---|---|---|---|---|
| | ✎ | Account Id | | OU_ID | |
| | ✎ | Active Status | | ACTIVE_FLG | |
| ✓ | ✎ | Address Name | | ADDR_NAME | |
| | ✎ | Address Name Locked | | NAME_LOCK_FLG | |

Figure 8. Business Address Business Component

## Graphical Representation of a Business Component and a Multi-Value Link

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

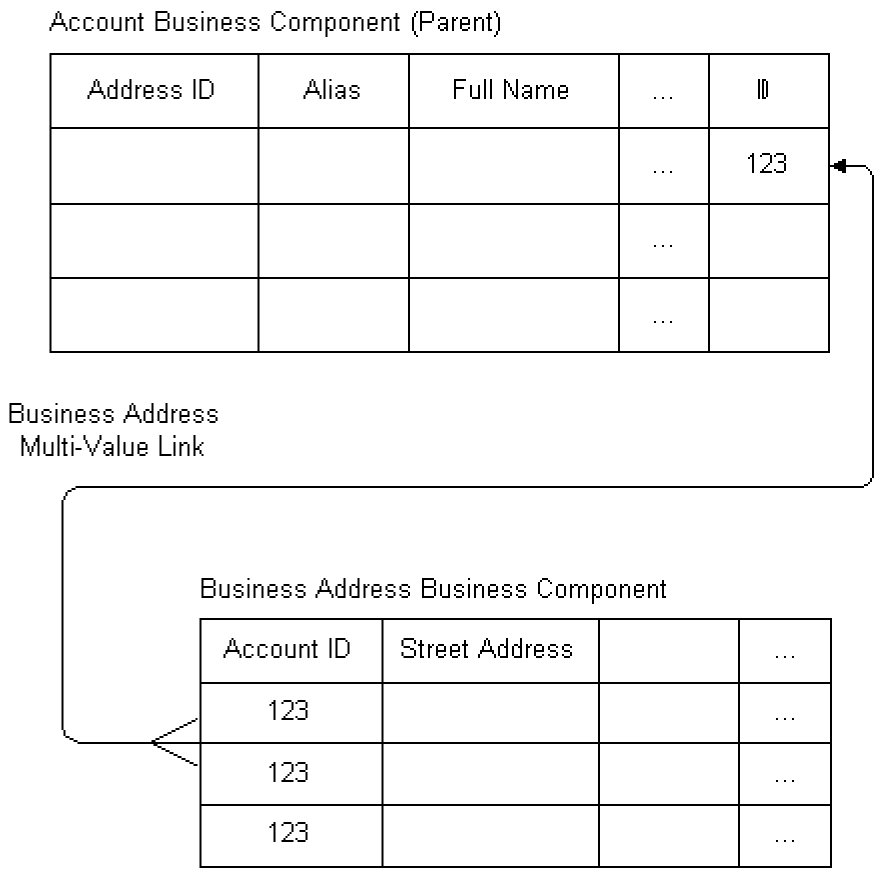

Account Business Component (Parent)
Address ID
Alias
Full Name
...
ID
123
Business Address Multi-Value Link
Business Address Business Component
Account ID
Street Address
123
123
123

Figure 9. Address Id Field and Business Address MVG

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

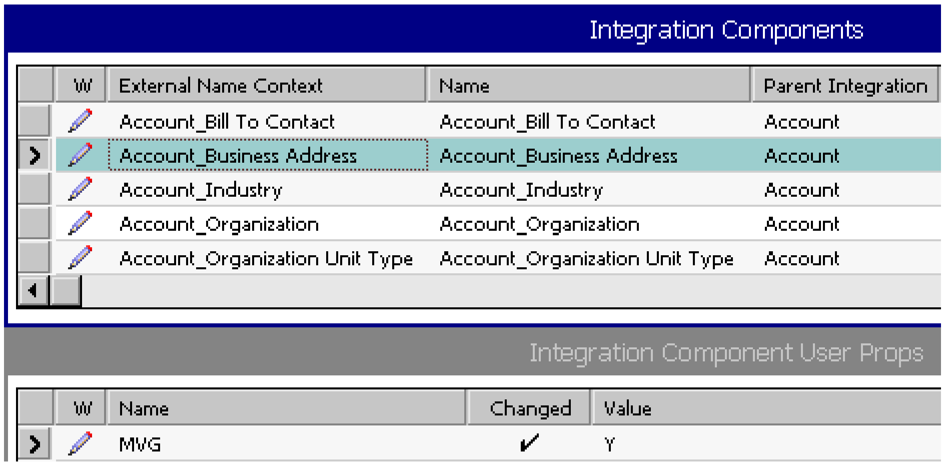
**NOTE:** Two business components are used to represent an MVG.

### Creating 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 Multi-value Link property and the Multi-value property are empty for these fields.
- The second integration component represents the MVG business component. In the example, this is the Business Address business component. The second integration component has one integration field for each field based on the given MVG in the parent business component. An integration component user property will be set on this integration component to tell the EAI Siebel Adapter that it is based on an MVG business component. If the MVG is a regular MVG, the user property is named *MVG*. If the MVG is an Association MVG, then the user property is named MVGAssociation. In both cases, the value of the user property is *Y*.

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



Integration Components

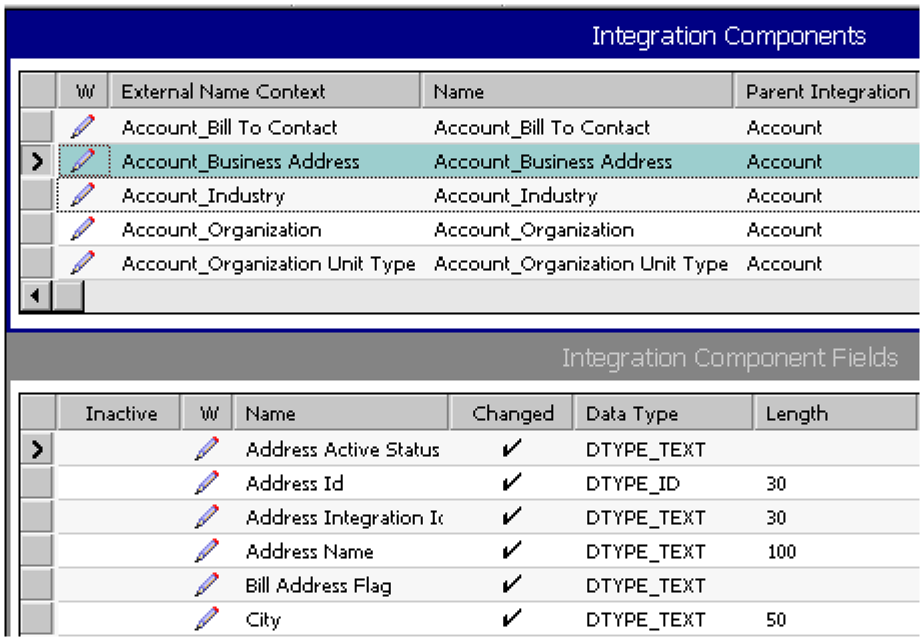
| | W | External Name Context | Name | Parent Integration |
|---|---|---|---|---|
| | | Account_Bill To Contact | Account_Bill To Contact | Account |
| > | | Account_Business Address | Account_Business Address | Account |
| | | Account_Industry | Account_Industry | Account |
| | | Account_Organization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

Integration Component User Props

| | W | Name | Changed | Value |
|---|---|---|---|---|
| > | | MVG | ✓ | Y |

Figure 10. Integration Component Based on MVG Business Component

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



| W | External Name Context | Name | Parent Integration |
|---|---|---|---|
| ✎ | Account_Bill To Contact | Account_Bill To Contact | Account |
| ✎ | Account_Business Address | Account_Business Address | Account |
| ✎ | Account_Industry | Account_Industry | Account |
| ✎ | Account_Organization | Account_Organization | Account |
| ✎ | Account_Organization Unit Type | Account_Organization Unit Type | Account |

| Inactive | W | Name | Changed | Data Type | Length |
|---|---|---|---|---|---|
| | ✎ | Address Active Status | ✓ | DTYPE_TEXT | |
| | ✎ | Address Id | ✓ | DTYPE_ID | 30 |
| | ✎ | Address Integration Id | ✓ | DTYPE_TEXT | 30 |
| | ✎ | Address Name | ✓ | DTYPE_TEXT | 100 |
| | ✎ | Bill Address Flag | ✓ | DTYPE_TEXT | |
| | ✎ | City | ✓ | DTYPE_TEXT | 50 |

Figure 11. Integration Component Fields Representing MVG

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

## 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, validation of the field can be done in EAI Siebel Adapter or Object Manager. To have the validation done in EAI Siebel Adapter, the integration component field should have a user property with the name PICKLIST and a value of Y; otherwise, validation is done by Object Manager.

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

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

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

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

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

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

If there is a field present in the business component and in the pickmap, and it is stored in the base table, then EAI Siebel Adapter can use the picklist to populate this field, only if this field is present and active in the integration component. This field should also be present and empty in the input property set.

## About Calculated Fields and Integration Objects

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

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

## 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, you need to activate them.

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

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

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

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

If the business requirement is to include the Project Name field, but not to limit the integration component's scope to only accounts with project, then you can change the join to S_PROJ in the Account business component to an outer join. For details on 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 Siebel Adapter if a corresponding operation is to be performed on an instance of that type. These properties take values Y or N.

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

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

## About Defining Field Dependencies

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

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

## About Setting Primaries Through Multi-Value Links

Primaries are set through multi-value links. However, you should not use multi-value links for modifying the linked component. To modify the linked component you should use links. If you need to set primaries in addition to modifying the linked component, use both links and multi-value links in your integration object. EAI Siebel Adapter should use the multi-value link only after it processes the component through the link; therefore, the link or the Association component should have a smaller external sequence number than the related MVG or MVGAssociation component. See "About the Structure of Integration Objects" on page 16 for an example.

## About Repository Objects

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

## About Integration Component Keys

There are multiple types of integration component keys.

- **User Key.** See "About User Keys" on page 28.
- **Status Key.** See "About Status Keys" on page 33.
- **Hierarchy Parent Key.** See "About Using the Hierarchy Parent Key" on page 34.
- **Hierarchy Root Key.** See "About Using the Hierarchy Root Key" on page 34.
- **Modification Key.** See "Configuring the EAI Siebel Adapter for Concurrency Control" on page 116.
- **Foreign Key.** See *Siebel Connector for Oracle Applications* .
- **Target Key.** See *Siebel Connector for Oracle Applications* .

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

## 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. You should make sure that the generated user keys match your business requirements; otherwise, inactivate them or add new user keys as appropriate.

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

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

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

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

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

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

**NOTE:** EAI Siebel Adapter uses user keys to match integration component instances with business component records. Since the match is case sensitive there is a chance that records are not matched if the case of the user key fields do not match. To avoid this, use the Force Case property on the business component field to make sure that user key fields are always stored in one case.

### About User Key Generation Algorithm

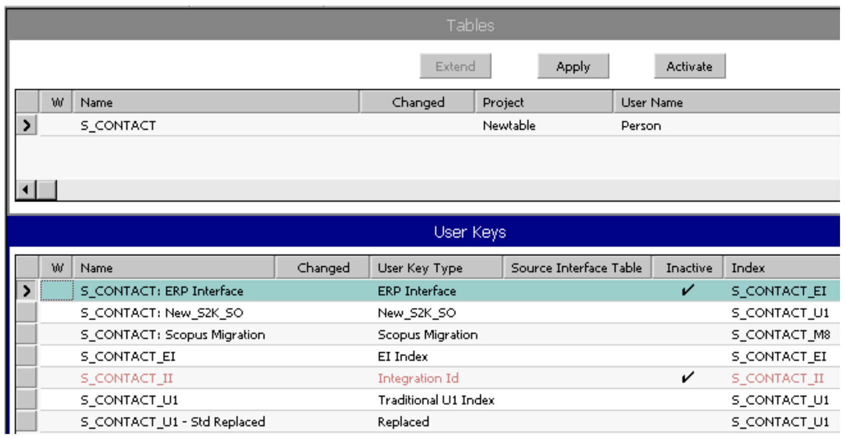
The Integration Object Builder wizard computes the user keys by traversing several Siebel objects, including the business object, business component, table, and link. This is because not every table user key meets the requirements to be used as the basis for integration object user keys.

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 12, the table S_CONTACT has several user keys.



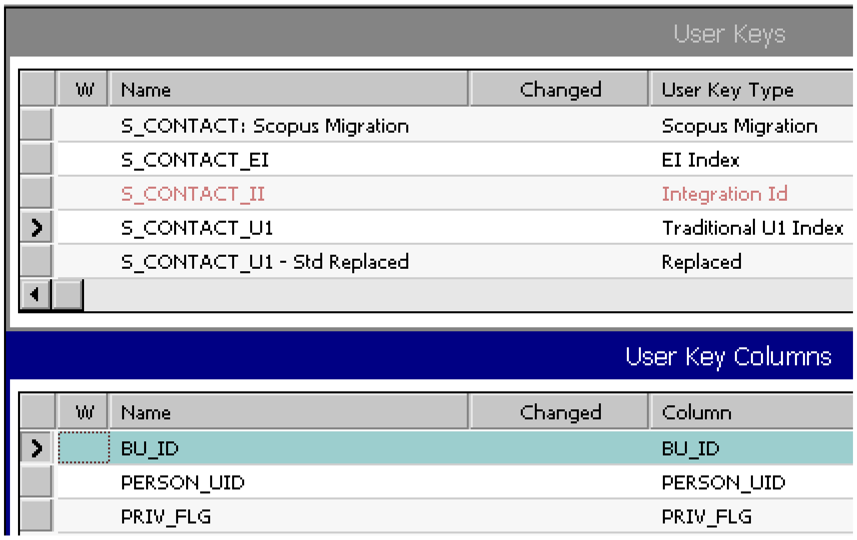
| W | Name | Changed | Project | User Name |
|---|---|---|---|---|
|  | S_CONTACT |  | Newtable | Person |

| W | Name | Changed | User Key Type | Source Interface Table | Inactive | Index |
|---|---|---|---|---|---|---|
|  | S_CONTACT: ERP Interface |  | ERP Interface |  | ✓ | S_CONTACT_EI |
|  | S_CONTACT: New_S2K_SO |  | New_S2K_SO |  |  | S_CONTACT_U1 |
|  | S_CONTACT: Scopus Migration |  | Scopus Migration |  |  | S_CONTACT_M8 |
|  | S_CONTACT_EI |  | EI Index |  |  | S_CONTACT_EI |
|  | S_CONTACT_II |  | Integration Id |  | ✓ | S_CONTACT_II |
|  | S_CONTACT_U1 |  | Traditional U1 Index |  |  | S_CONTACT_U1 |
|  | S_CONTACT_U1 - Std Replaced |  | Replaced |  |  | S_CONTACT_U1 |

Figure 12. User Keys for Table S_CONTACT

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

Each User Key Column defined for a given user key must be exposed to the business component in which you are interested. For example, Figure 13 shows three user key columns for the user key S_CONTACT_U1.


User Keys

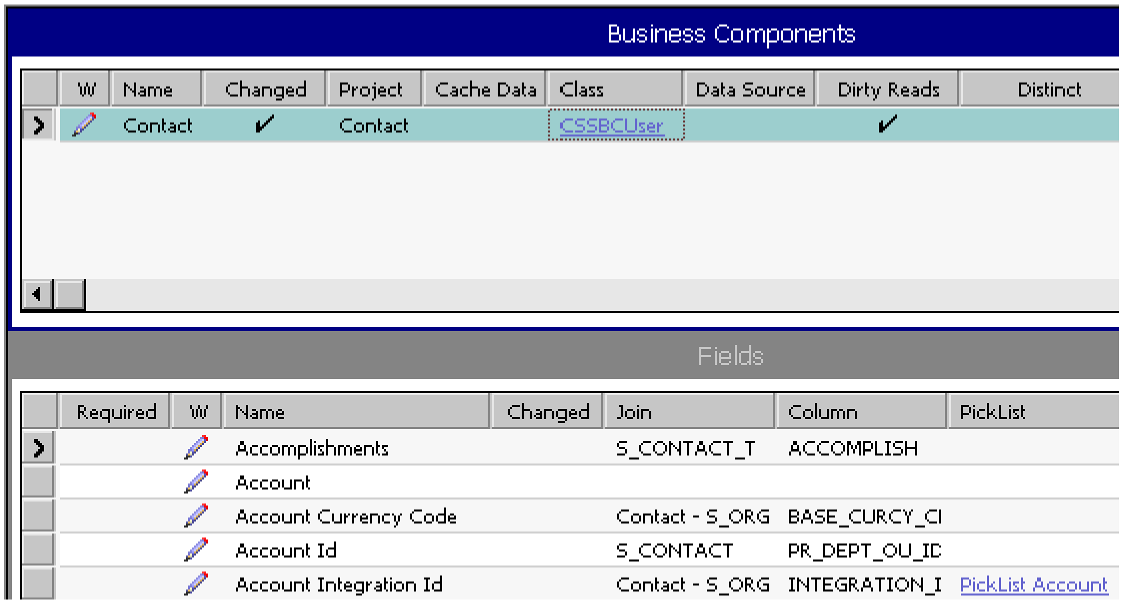
| W | Name | Changed | User Key Type |
|---|---|---|---|
| | S_CONTACT: Scopus Migration | | Scopus Migration |
| | S_CONTACT_EI | | EI Index |
| | S_CONTACT_II | | Integration Id |
| | S_CONTACT_U1 | | Traditional U1 Index |
| | S_CONTACT_U1 - Std Replaced | | Replaced |

User Key Columns

| W | Name | Changed | Column |
|---|---|---|---|
| | BU_ID | | BU_ID |
| | PERSON_UID | | PERSON_UID |
| | PRIV_FLG | | PRIV_FLG |

Figure 13. User Key Columns for the S_CONTACT_U1 User Key

If the columns of the user key are exposed in the business component and those columns are not foreign keys, the Integration Object Builder wizard creates an integration component key based on the table’s user key. The Integration Object Builder wizard also defines one integration component key field corresponding to each of the table’s user key columns. For example, in Figure 14, the user key columns are exposed in the Integration Component Fields applet for the Contact integration component.

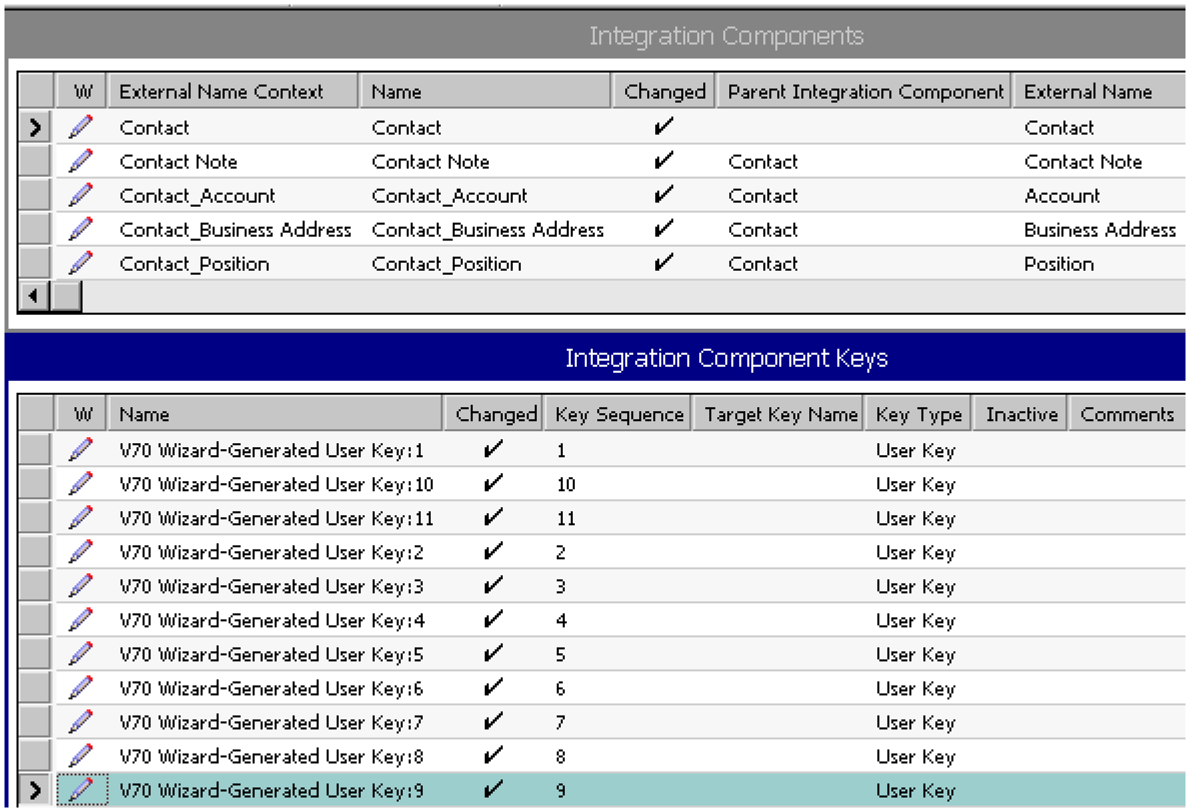


| W | Name | Changed | Project | Cache Data | Class | Data Source | Dirty Reads | Distinct |
|---|---|---|---|---|---|---|---|---|
|  | Contact | ✓ | Contact |  | CSSBCUser |  | ✓ |  |

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

Figure 14. Integration Component Field List

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



Integration Components

| | W | External Name Context | Name | Changed | Parent Integration Component | External Name |
|---|---|---|---|---|---|---|
| > | | Contact | Contact | ✓ | | Contact |
| | | Contact Note | Contact Note | ✓ | Contact | Contact Note |
| | | Contact_Account | Contact_Account | ✓ | Contact | Account |
| | | Contact_Business Address | Contact_Business Address | ✓ | Contact | Business Address |
| | | Contact_Position | Contact_Position | ✓ | Contact | Position |

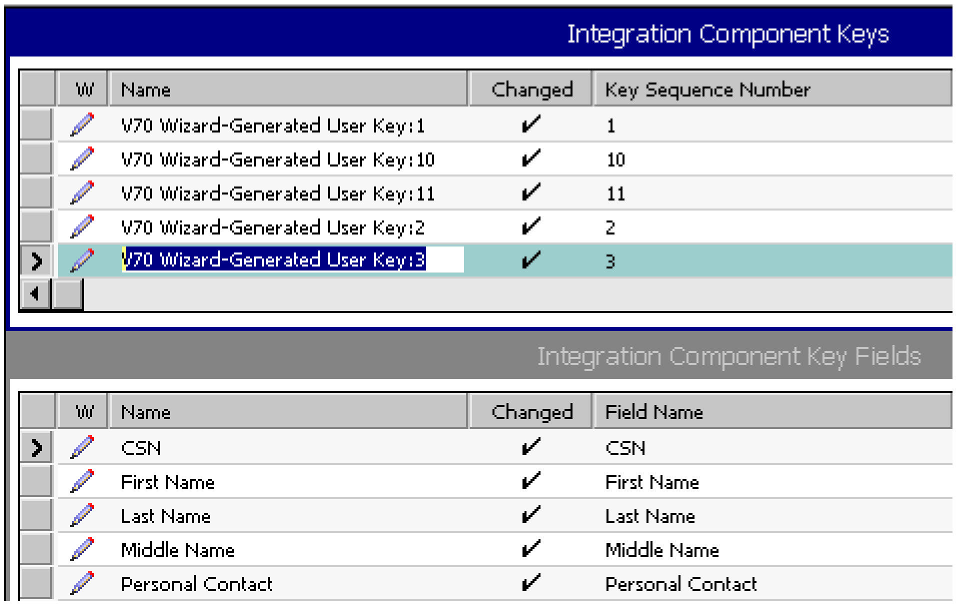
Integration Component Keys

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

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

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

**NOTE:** You should only modify user keys if you have a good understanding of the business component and integration logic.



Integration Component Keys

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

Integration Component Key Fields

| W | Name | Changed | Field Name |
|---|---|---|---|
| ✎ | CSN | ✓ | CSN |
| ✎ | First Name | ✓ | First Name |
| ✎ | Last Name | ✓ | Last Name |
| ✎ | Middle Name | ✓ | Middle Name |
| ✎ | Personal Contact | ✓ | Personal Contact |

Figure 16. Contact Integration Component Key Fields

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

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

## 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 EAI Siebel Adapter to True to return an integration object instance as a status object.

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

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

## 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 very first quote item inserted by EAI Siebel Adapter has the Parent Id set to blank, but for each child quote item, EAI Siebel Adapter checks the keys to figure out which fields are to be set. If Hierarchy Parent Key is not defined, then the child quote item is inserted as a new quote item without a link to its parent (denormalized).

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

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

## 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 EAI Siebel Adapter to successfully perform an operation, that operation needs to be allowed at all levels. If the operation is allowed at every level but the field level, a warning message is logged in the log file and processing continues. Otherwise, an error message is returned and the transaction is rolled back. The permission settings are done by using user properties on the integration object, component, or field.

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

Table 5. Permission Flags for an Integration Component

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

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

## About the EAI Siebel Adapter Access Control

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

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

**NOTE:** For details on ViewMode, see *Siebel Tools Online Help*.

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

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

| | **Checklist** |
|---|---|
| ❑ | Create integration objects using the EAI Siebel Wizard.<br>For details, see "Creating Integration Objects Using the EAI Siebel Wizard" on page 38. |
| ❑ | Fine-tune your integration object.<br>For details, see "Siebel Integration Object Fine-Tuning" on page 40. |
| ❑ | Validate your integration object.<br>For details, see "Integration Object Validation" on page 40. |

## Creating Integration Objects Using the EAI Siebel Wizard

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

#### *To create a new Siebel integration object*

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

      **NOTE:** The name of an integration object must be unique among other integration objects. 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 would like the wizard to ignore. This means you will not be able to integrate data for that component between the Siebel application and another system.

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

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

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

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

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

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

**NOTE:** After creating integration objects in Siebel Tools, you must compile 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:** Once you create your integration object based on a Siebel business object, you should not change its integration component's External Name Context; otherwise, the synchronization process will not recognize the integration component and will remove it from the integration object.

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

The Integration Component Fields applet displays the list of fields for that component. Note the system fields Conflict Id, Created, Id, Mod Id, Updated, operation, and searchspec in the list. This setting prevents EAI Siebel Adapter Query and QueryPage method from outputting these fields. For more details, see "About the SearchSpec Input Method Argument" on page 112.

### 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 will 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 very 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 such that it will not affect any business logic.

## Siebel Integration Object Fine-Tuning

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

- Deactivate the fields that do not apply to your business requirements.
- If necessary, activate the fields that have been deactivated by the Siebel wizard. For details, see Chapter 2, “Integration Objects.”
- Add the fields that have not been included by the Siebel wizard. For details on the implications of activating such fields, see “About Calculated Fields and Integration Objects” on page 25 and “About Inner Joins and Integration Components” on page 25.
- Validate the user keys. For details, see Chapter 2, “Integration Objects.”
- Update the user properties for your integration object to reflect your business requirements. For details, see “Resolving Synchronization Conflicts for Integration Objects and User Properties” on page 51.

## Integration Object Validation

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

***To validate your integration object***

1. Start Siebel Tools.
2. Select your integration object.
3. Click Validate.
4. Review the report and modify your integration object as needed.

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

## Integration Objects Synchronization

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

Some examples of these changes are:

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

### Synchronizing Integration Objects

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

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

| **Checklist** | |
|---|---|
| ❑ | Run the Synchronization wizard.<br><br>For details, see "Updating the Entire Integration Object" on page 44. |

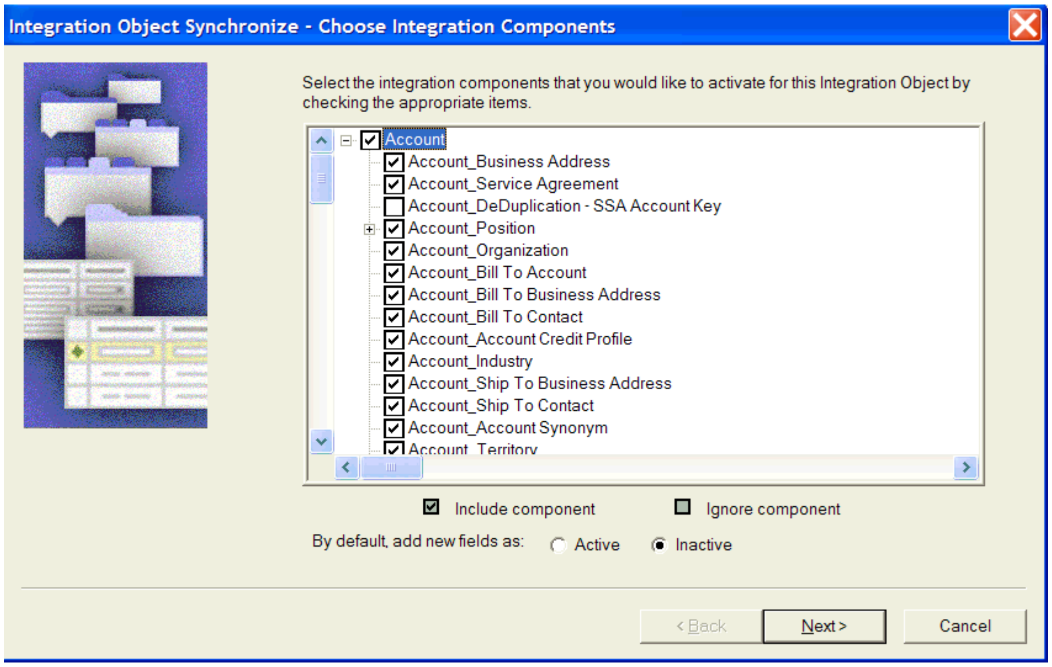
| | Checklist |
|---|---|
| ❑ | Modify the newly updated integration object as needed, using the DIFF tool and a copy of the original integration object as reference.<br><br>For details, see *Using Siebel Tools*. |
| ❑ | Run Validation.<br><br>For details, see "Integration Object Validation" on page 40. |

***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.



Integration Object Synchronize - Choose Integration Components

Select the integration components that you would like to activate for this Integration Object by checking the appropriate items.

- [✓] Account
  - [✓] Account_Business Address
  - [✓] Account_Service Agreement
  - [ ] Account_DeDuplication - SSA Account Key
  - [✓] Account_Position
  - [✓] Account_Organization
  - [✓] Account_Bill To Account
  - [✓] Account_Bill To Business Address
  - [✓] Account_Bill To Contact
  - [✓] Account_Account Credit Profile
  - [✓] Account_Industry
  - [✓] Account_Ship To Business Address
  - [✓] Account_Ship To Contact
  - [✓] Account_Account Synonym
  - [✓] Account_Territory

☑ Include component ☐ Ignore component

By default, add new fields as: Active [✓ Inactive]

   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 details, 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.

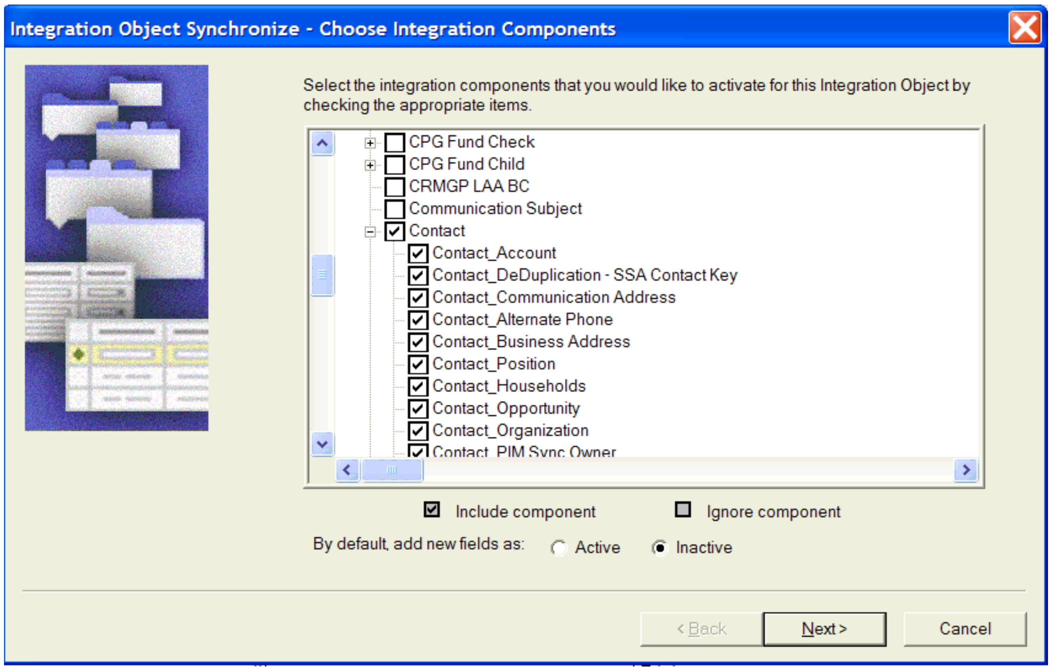
4. Deactivate any new fields that you do not need in a component of your updated integration object.

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

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

## Synchronization Rules

During the synchronization process, the Synchronization wizard follows particular update rules. Consider a simple example involving the Siebel Account integration object with only Contact and its child components marked as active in the object. Figure 17 helps you to visualize this example.


Integration Object Synchronize - Choose Integration Components

Select the integration components that you would like to activate for this Integration Object by checking the appropriate items.

- [ ] CPG Fund Check
- [ ] CPG Fund Child
- [ ] CRMGP LAA BC
- [ ] Communication Subject
- [x] Contact
  - [x] Contact_Account
  - [x] Contact_DeDuplication - SSA Contact Key
  - [x] Contact_Communication Address
  - [x] Contact_Alternate Phone
  - [x] Contact_Business Address
  - [x] Contact_Position
  - [x] Contact_Households
  - [x] Contact_Opportunity
  - [x] Contact_Organization
  - [x] Contact_PIM Sync Owner

☑ Include component ☐ Ignore component

By default, add new fields as: Active [✓ Inactive]

Figure 17. Example of Selected Integration Components

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

### Updating the Entire Integration Object

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

Figure 18 illustrates this concept.

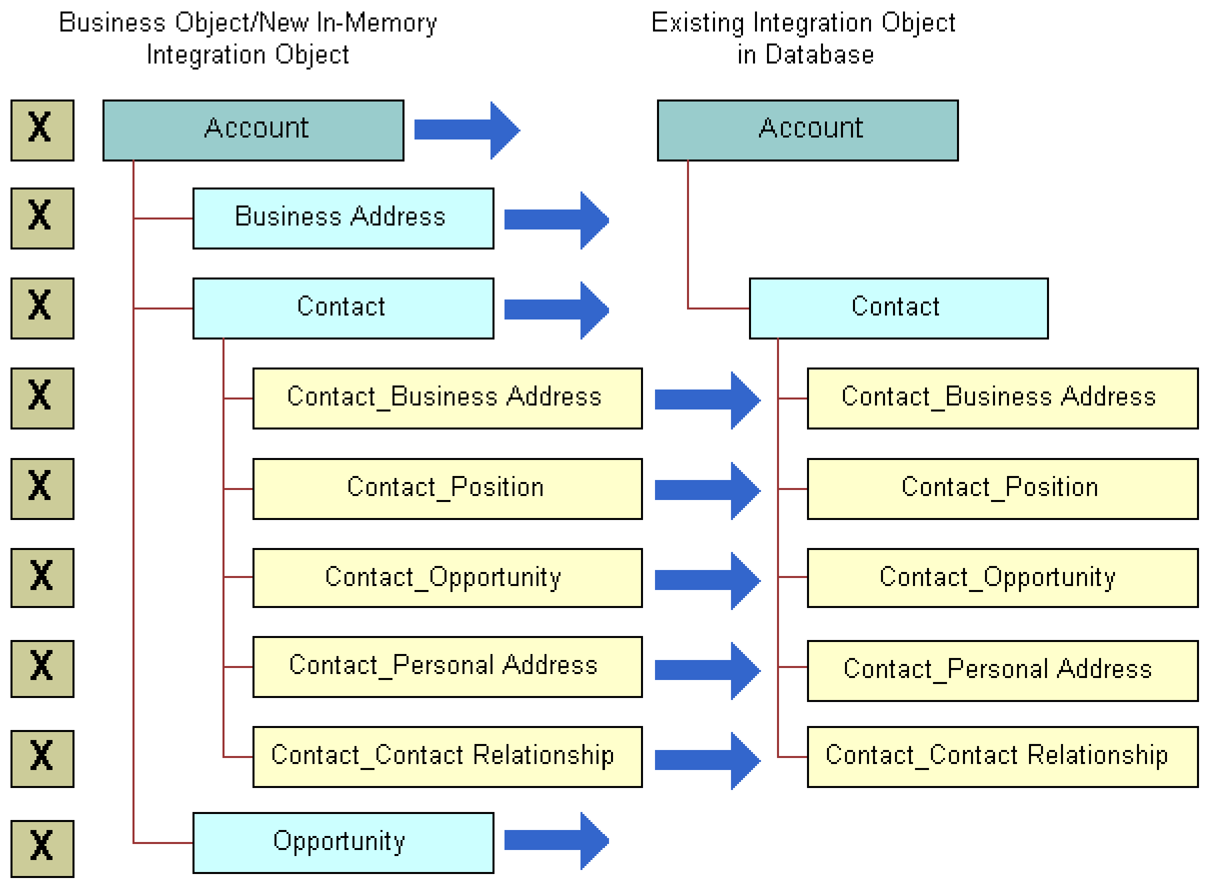


Figure 18. Synchronizing the Integration Object

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

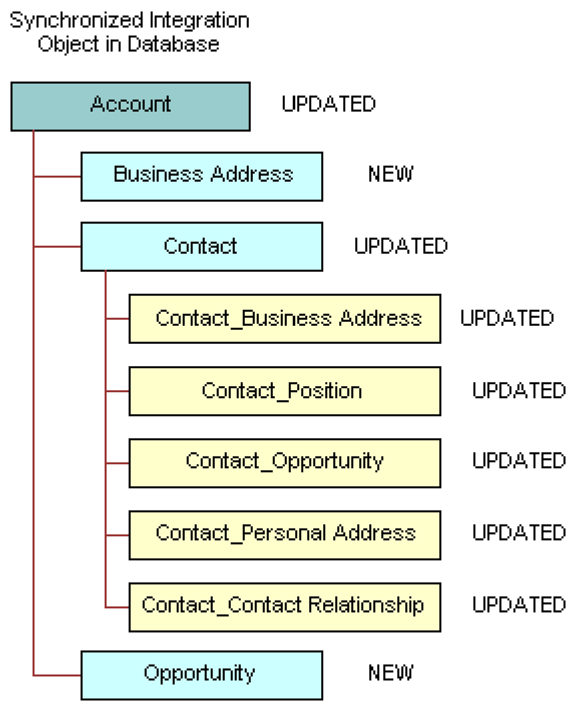


Figure 19. Completely Updated Integration Object

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

## About Deleting a Component from the Integration Object

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

In Figure 20 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 20 illustrates this concept.

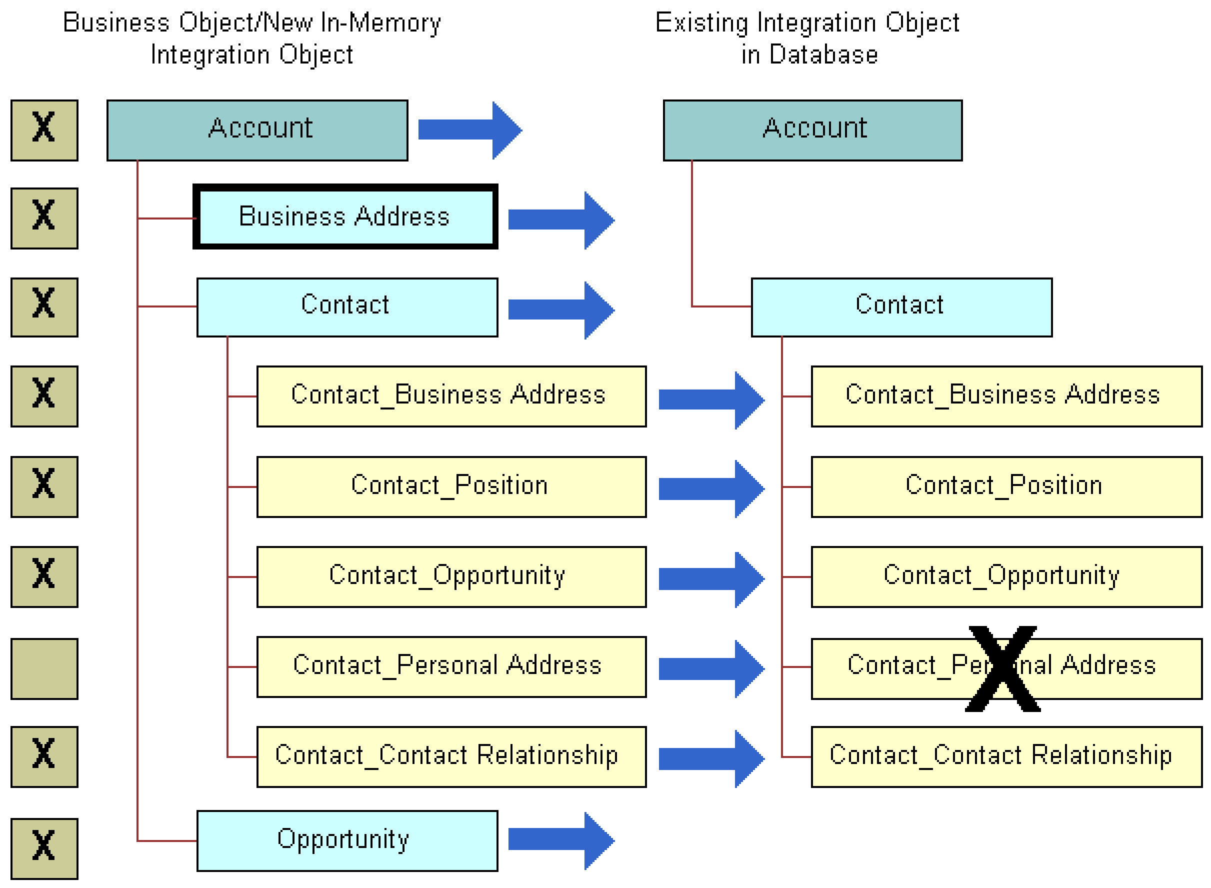


Figure 20. Deleting a Component from the Integration Object

Figure 21 shows the integration object after synchronization.

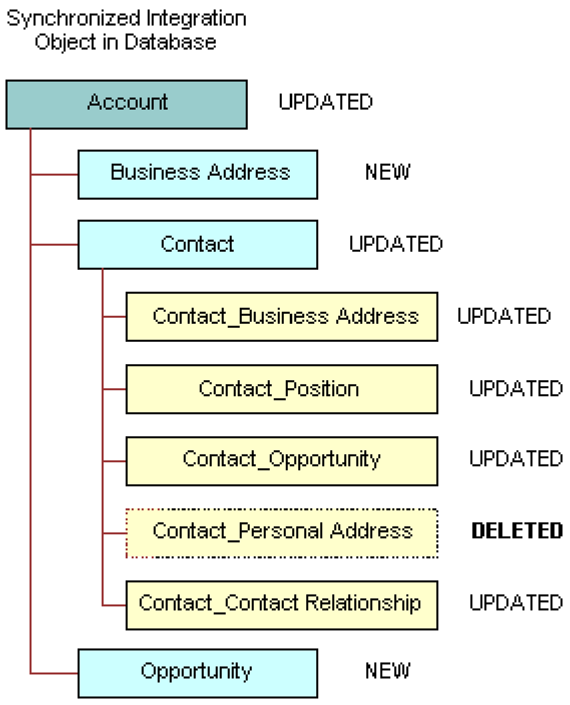


Figure 21. Synchronization Resulting in a Deleted Component

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

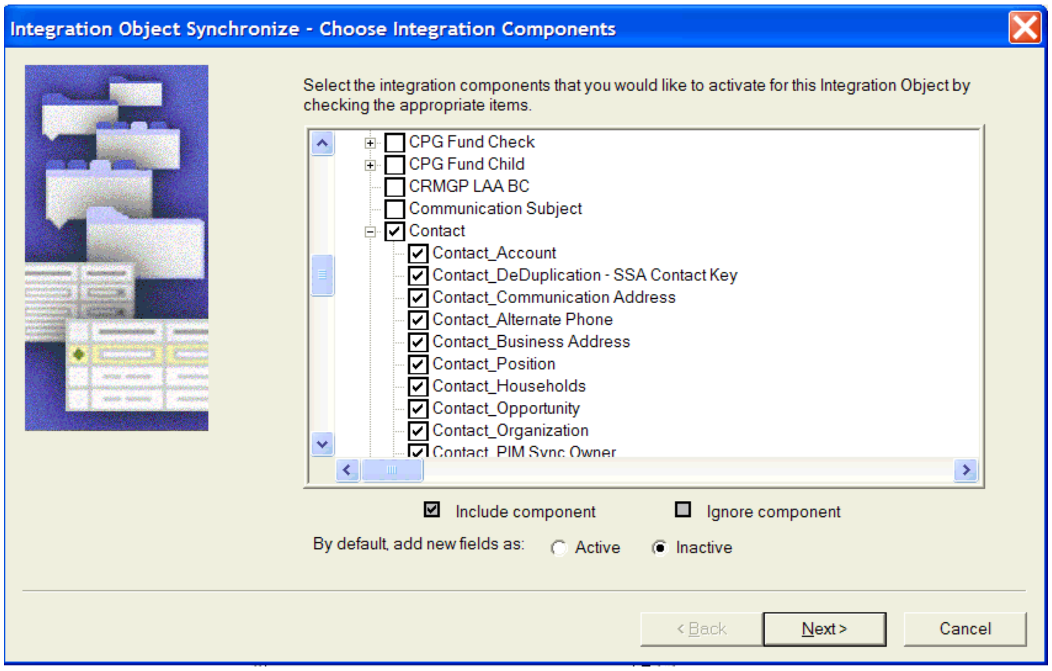
This example is intended to show how you might cause unexpected results by deselecting components. However, if you do want to delete a particular component from the integration object, deleting a component from the integration object method accomplishes that goal.

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

## 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 22, the wizard returns a list of the available business objects from which you can choose one to base your integration object on.

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


Integration Object Synchronize - Choose Integration Components

Select the integration components that you would like to activate for this Integration Object by checking the appropriate items.

- ☐ CPG Fund Check
- ☐ CPG Fund Child
- ☐ CRMGP LAA BC
- ☐ Communication Subject
- ☑ Contact
  - ☑ Contact_Account
  - ☑ Contact_DeDuplication - SSA Contact Key
  - ☑ Contact_Communication Address
  - ☑ Contact_Alternate Phone
  - ☑ Contact_Business Address
  - ☑ Contact_Position
  - ☑ Contact_Households
  - ☑ Contact_Opportunity
  - ☑ Contact_Organization
  - ☑ Contact_PIM Sync Owner

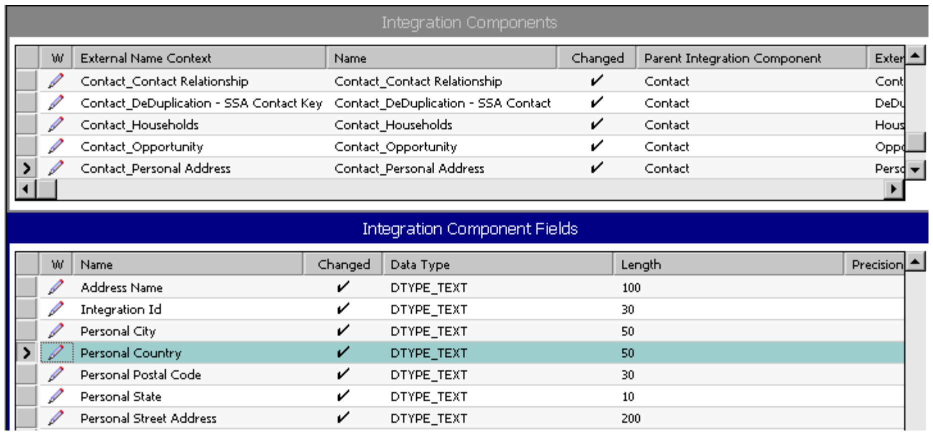
☑ Include component ☐ Ignore component

By default, add new fields as: Active [✓ Inactive]

Figure 22. Activated Components in the Contact Integration Object

After the wizard creates your integration object, you can edit the object in Siebel Tools, as shown in Figure 23. You might choose to drill down into the integration components and activate or inactivate particular components or even particular fields within one or more components.

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


Integration Components

| W | External Name Context | Name | Changed | Parent Integration Component | Exter |
|---|---|---|---|---|---|
| ✎ | Contact_Contact Relationship | Contact_Contact Relationship | ✓ | Contact | Cont |
| ✎ | Contact_DeDuplication - SSA Contact Key | Contact_DeDuplication - SSA Contact | ✓ | Contact | DeDu |
| ✎ | Contact_Households | Contact_Households | ✓ | Contact | Hous |
| ✎ | Contact_Opportunity | Contact_Opportunity | ✓ | Contact | Oppo |
| ✎ | Contact_Personal Address | Contact_Personal Address | ✓ | Contact | Perso |

Integration Component Fields

| W | Name | Changed | Data Type | Length | Precision |
|---|---|---|---|---|---|
| ✎ | Address Name | ✓ | DTYPE_TEXT | 100 | |
| ✎ | Integration Id | ✓ | DTYPE_TEXT | 30 | |
| ✎ | Personal City | ✓ | DTYPE_TEXT | 50 | |
| ✎ | Personal Country | ✓ | DTYPE_TEXT | 50 | |
| ✎ | Personal Postal Code | ✓ | DTYPE_TEXT | 30 | |
| ✎ | Personal State | ✓ | DTYPE_TEXT | 10 | |
| ✎ | Personal Street Address | ✓ | DTYPE_TEXT | 200 | |

Figure 23. Activated Components in the Contact Integration Object

## Siebel Integration Objects Maintenance and Upgrade

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

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

- Name any user key that you add differently from the generated user keys. Using meaningful names helps with debugging.
- Inactivate user keys instead of deleting them.
- Inactivate fields instead of deleting them.

## Resolving Synchronization Conflicts for Integration Objects and User Properties

This section serves as a guide to resolving synchronization conflicts should any arise.

Table 6 illustrates behavior of the merging logic for each of the integration object parts that have to be synchronized.

Table 6. 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 current integration object are checked in the UI tree, other components presented as Inactive. User decides which components to add/ delete. This is done by the Synchronization wizard UI. |
| Fields | Keep 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 info).<br><br>System fields are created when appropriate (for example searchspec, IsPrimaryMVG, and operation). If system field is inconsistent with integration component definition, delete it.<br><br>Active/Inactive - Preserve 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 current integration object values to keep XML compatible. Add new components/fields properties avoiding conflict with existing XML.<br><br>XML Properties are processed according to the XML sequence. New components/fields XML sequence within the parent component element will be higher than current.<br><br>Existing processing code should be reused (and checked for correct behavior). |

Table 6. Merging Logic Used for Synchronizing Integration Objects

| Integration Object Metadata | Merging Rules |
|---|---|
| External Sequence on components/fields | Preserve component/field sequence within the parent component. Set sequence on new components/fields higher than existing ones. |
| Name | Preserve Names in the current integration object. |
| User key, Hierarchy key, Other keys (for example, Status Key) | Existing Keys:<br>■ Keep existing keys as Active if all the key fields are Active.<br>■ Make Inactive if Inactive already or if any of the fields are Inactive.<br>■ If a field is Inactive in an integration component, make it Inactive in the key. Make the key Inactive.<br>■ If a field is not present in an integration component, delete it from the key. Make the key Inactive.<br>New Keys:<br>■ Create new keys as Inactive.<br>■ If any of the key fields are Inactive, either:<br>  ■ Do not create the key.<br>  ■ Make fields Active in the integration component. |
| User Properties | Preserve valid cases, remove invalid ones and generate warnings. |

Table 7 shows the logic that will be used when synchronizing user properties.

Table 7. Logic Used for Synchronizing User Properties

| **User Property Name** | **Values (Default is in *italics*)** | **Level (*Object*, *Component*, or *Field*)** | **Merging Rules** |
|---|---|---|---|
| Association | Y,*N* | C | Siebel Wizard generates the value based on current business component definition. Wizard overwrites user change since in order for 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 current business component definition. Wizard overwrites user change since in order for integration component to be functional, the User Property has to be consistent with the business component.<br><br>IsPrimaryMVG system field is created in merged integration object. |
| Picklist | Y,*N* | F | Siebel Wizard generated, user change is kept if valid (if Picklist component).<br><br>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 = Y on integration component.<br><br>If no Active fields left, remove the user property. |
| IgnoreBoundedPicklist | Y,*N* | O, C, F | Entered by user, keep if valid (if component Picklist = Y). |

Table 7. Logic Used for Synchronizing User Properties

| **User Property Name** | **Values (Default is in *italics*)** | **Level (*Object*, *Component*, or *Field*)** | **Merging Rules** |
|---|---|---|---|
| MVGAssociation | Y,*N* | C | Siebel Wizard generates the value based on current business component definition. Wizard overwrites user change since in order for integration component to be functional, the User Property has to be consistent with the business component.<br><br>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. Wizard overwrites user change since in order for integration component to be functional, the User Property has to be consistent with the business component. (component MVG = Y) |
| AssocFieldName | Any valid field name in the Association business component. | F | Siebel Wizard generates the value based on current business component definition. Wizard overwrites user change since in order for integration component to be functional, the User Property has to be consistent with the business component. (component MVGAssociation = Y) |
| NoInsert, NoDelete, NoUpdate, NoQuery, NoSynchronize | Y,*N* | C, F (NoUpdate) | Entered by the user. Keep the current value. |

Table 7. Logic Used for Synchronizing User Properties

| **User Property Name** | **Values (Default is in *italics*)** | **Level (*Object*, *Component*, or *Field*)** | **Merging Rules** |
|---|---|---|---|
| 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. Keep the current value. |
| ViewMode | *All*, Manager, Sales Represent, and any others | O | Entered by the user. Keep the current value. |
| 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

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

7 From the list of components, select Contact and Opportunity. There is also a component named Contact_Opportunity in the list. This component is an MVGAssociation component, and should be picked only if you need this integration object to set the primary opportunity for contact. For details on MVG, see "About MVGs in EAI Siebel Adapter" on page 111.

8 Deactivate all integration component fields in the Contact integration component except First Name, Last Name, Login Name, and Comment. (In this example, these are the only fields we need for Contact.)

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

10 Compile a new SRF file and copy the SRF file to the *SIEBSRVR_ROOT*/OBJECTS directory.

### *To test the newly created integration object*

1 Start 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 in "To create an integration object with a many-to-many business component" on page 55.

4 Run the workflow processes using the Workflow Process Simulator.

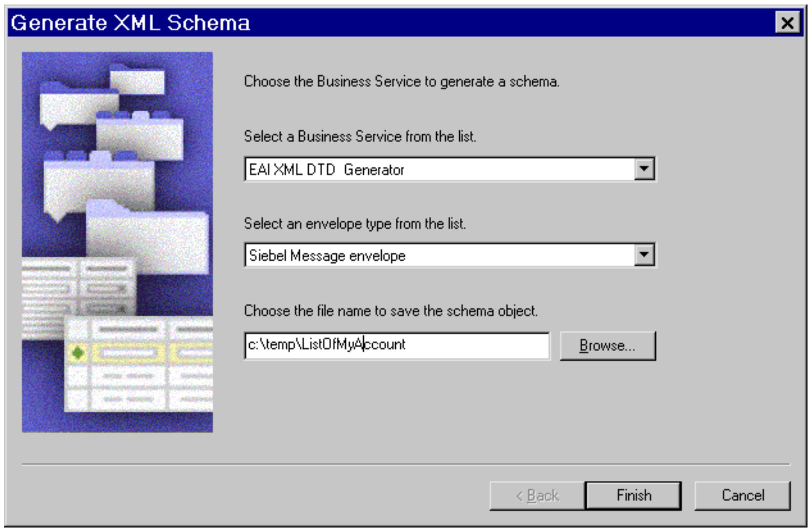
## Generating Integration Object Schemas

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 shown in the following figure.



Choose the Business Service to generate a schema.

Select a Business Service from the list.
EAI XML DTD Generator

Select an envelope type from the list.
Siebel Message envelope

Choose the file name to save the schema object.
c:\temp\ListOfMyAccount

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 EAI Siebel Adapter operations. You should inactivate unnecessary fields and components in your integration objects.

### Force-Active Fields

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

### Picklist Validation

Siebel applications have two classes of picklists—static picklists based on 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, validation in the adapter can be turned off because the Object Manager can perform the validation. Turning off the validation at the EAI Siebel Adapter level means that picklist-related warnings and debugging messages will not show up along with other EAI Siebel Adapter messages. This also means that bounded picklist errors will not be ignored, even if Ignore Bounded Picklist is set to Y.

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 especially occurs when the picklist is based on an MLOV. If the EAI Siebel Adapter is performing the validation (PICKLIST = 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 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, Internal Product can only be modified by an administrator. The same restrictions apply during integration. In many cases, the Siebel Integration Object Builder wizard detects the restrictions and sets properties such as No Insert or No Update on the integration components.

### System Fields

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

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

## Best Practices 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. See "About Optimizing Performance for Using Integration Objects" on page 57.
- Design with maintenance in mind. See "Siebel Integration Objects Maintenance and Upgrade" 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 EAI Siebel Adapter to change the Auto Primary property according to the input request, while making sure that there is always a primary child selected.

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

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

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

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

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

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

# 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.

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

### Creating Business Services

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

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

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

- **In Siebel Tools.** Created at design time in Siebel Tools using Siebel VB or Siebel eScript. Design-time business services are stored in the Siebel repository (*.srf), so you have to compile the repository before testing them. Once your test is completed, you need to compile and disseminate the SRF to your clients. The business services stored in the repository will automatically come over to the new repository during the upgrade process. General business services are based on the class CSSService*;* however, for the purposes of Siebel eAI, you base your data transformation business services on the CSSEAIDTEScriptService class. For details, 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 details, see "Creating a Business Service in the Siebel Client" on page 68.

**NOTE:** To use the DTE scripts, you need to 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 details 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 24 illustrates the concept of a property set.

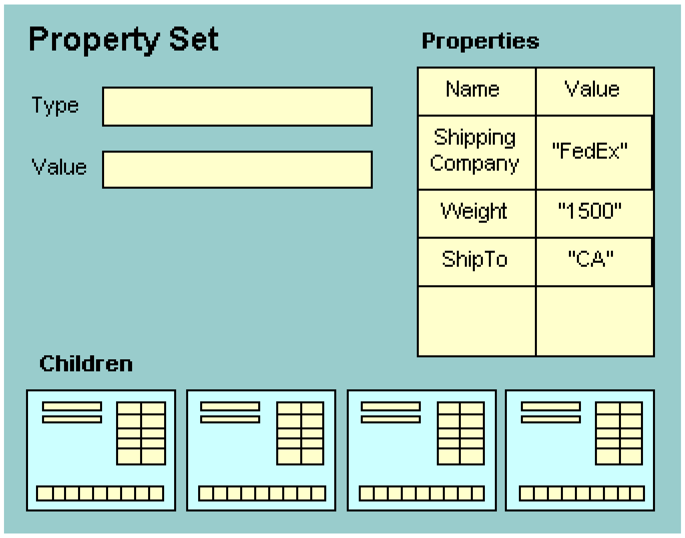


Figure 24. Property Set Structure

The property set consists of four parts:

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

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

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

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

## Creating Business Services in Siebel Tools

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

| | **Checklist** |
|---|---|
| ❑ | Define the Business Service<br>For details, see "To define a business service in Siebel Tools" on page 64. |
| ❑ | Define the Business Service Methods<br>For details, see "To define a business service method" on page 65. |
| ❑ | Define the Business Service Methods Arguments<br>For details, see "To define the business service method arguments" on page 65. |
| ❑ | Define Business Service Scripts<br>For details, see "To define and write the business service script" on page 66. |
| ❑ | Define Business Service Subsystem<br>For details, see "To specify a business service subsystem" on page 67. |
| ❑ | Define Business Service User Properties<br>For details, see "To define business service user properties" on page 67. |

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

## Defining a Business Service in Siebel Tools

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

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

1 Start Siebel Tools.

2 Select and lock the project you want to associate your business service with.

   **NOTE:** Each business service must belong to a project and the project must be locked. For details, see *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 should 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 details 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, you need to specify the Display Name as well.
4. Enter the data type in the Data Type field.

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

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

## Defining and Writing Business Service Scripts

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

### *To define and write the business service script*

1. Start Siebel Tools.
2. Select the business service for which you want to write a script.
3. Right-click to display a pop-up menu.
4. Choose Edit Server Scripts.
5. Select either 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:** You need to write your business service in Siebel eScript if you want to use the DTE scripts. For details on scripting, see *Using Siebel Tools*.

## Specifying Business Service Subsystems

You can optionally specify a business service subsystem. A business service subsystem is a server component that encapsulates a large amount of functionality and that is already included in the Siebel repository. Business service subsystems define particular events upon which the subsystem will be called. The subsystems can also trigger other events, depending on how they are defined. Examples of business service subsystems are presented in Table 8.

Table 8. Business Service Subsystems

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

### *To specify a business service subsystem*

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

## Defining Business Service User Properties

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

### *To define business service user properties*

1. With your business service selected, double-click the Business Service User Prop folder in the Tools Object Explorer to display the list of Business Service User Props.
2. Choose Edit > New Record to create a blank user property record.
3. Type the name of the user property in the Name field.

4 Type a value in the Value field.

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

## Creating a Business Service in the Siebel Client

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

### *To define a business service in the Siebel Client*

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

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 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

Business services can be exported 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. From the application-level menu, choose Navigate > Site Map > Administration - Business Service > Simulator.

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

2. In the 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:** Once 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 produce a property set as an output. The output property set is used by EAI Siebel Adapter to insert a new account into the Siebel application:

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

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

var child = TheApplication().NewPropertySet();

var psInputs = TheApplication().NewPropertySet();

var psOutputs = TheApplication().NewPropertySet();

var psOutputs2 = TheApplication().NewPropertySet();

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

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

psOutputs.SetType "SiebelMessage";

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

psOutputs.SetProperty "MessageId", "";

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

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

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

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

```
Set Inp = TheApplication.NewPropertySet

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

Inp.SetProperty "DispatchService", "XML Converter"

Inp.SetProperty "DispatchMethod" , "XMLToPropSet"

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

Set XMLOutputs = theApplication.NewPropertySet

svc.InvokeMethod "ReceiveDispatch", Inp, XMLOutputs

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 you need to write to create the property set may look something like this:

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

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

var inputs; // variable to contain the XML input

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

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

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

   outputs = TheApplication().NewPropertySet();

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

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

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

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

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

# 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). Web Services can be reused regardless of how they are implemented. Web Services can be developed on any computer platform and in any development environment as long as they can communicate with other Web Services using these common protocols.

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

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

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

### Supported Web Services Standards

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

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

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

## About RPC-Literal and DOC-Literal Bindings

By providing the ability to publish a Siebel Web Service as a Document-Literal or RPC-Literal bound Web Service, the Siebel application is conforming to the assertion as defined by the Web Services Interoperability Organization's (WS-I) Basic Profile specification. Adherence to this specification will make sure that the Siebel application can interoperate with external Web Service providers.

**NOTE:** 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) since we do not have to use SOAPAction header to specify the operation.

The following assertions 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 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 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 screen.

#### *To make a Web Service an RPC-literal Web Service*

1. From the application-level menu, choose Navigate > Site Map > Administration - Web Services > Inbound Web Services.
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

In a document-literal SOAP binding, the serialized element child of the soap:Body gets its namespace from the targetNamespace of the schema that defines the element. Use of the namespace attribute of the soapbind:body element would override the element's namespace.

**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. Release 7.7 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.

One-way operations should be considered for usage 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.

### About Defining Support for One-Way Operations

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

- **Specification R2714.** For an 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. Once 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. Once 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.

### *To create an Inbound Web Service record*

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

1. From the application-level menu, choose Navigate > Site Map > Administration - Web Services > Inbound Web Services.
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 MQSeries AMI or EAI MSMQ Server transports, enter one of the following:

  `mq://`*`YourQueueName@YourQueueManagerName`*

  `msmq://`*`YourQueueName@YourQueueMachineName`*

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

**i** Select the binding that will publish the Web Service.

**NOTE:** Release 7.7 supports RPC_Encoded, RPC_Literal, and DOC_Literal styles of binding.

**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.

Once you have created a new Inbound Web Service record you can generate a WSDL document, as described in the following procedure.

### *To generate a WSDL file*

1. 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). It 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 can then be used 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 utilize.

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:

1. A WDSL file for the external Web Service.

or

2. 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 complex arguments are not used by the service, 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 details, 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 screen.

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

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

### *To manually create a new outbound Web Service*

1. From the application-level menu, choose Navigate > Site Map > Administration - Web Services > 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 over 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 over 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 will vary 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 over 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:** Release 7.7 supports RPC_Encoded, RPC_Literal, DOC_Literal, and Property Set styles of binding.

Property Set Binding should be used when the input Property Set to the proxy service is forwarded without changes to the destination address. This is intended primarily for use in combination with Local Workflow or Local Business Service transport to avoid overhead of processing XML.

**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 93.

**5** Generate the WSDL file. For details, see "To generate a WSDL file" on page 78.

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

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

1. 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.

  You need to change the output from your business services in Siebel Tools, as well as the name of the property set child that contains 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 utilize 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 25.

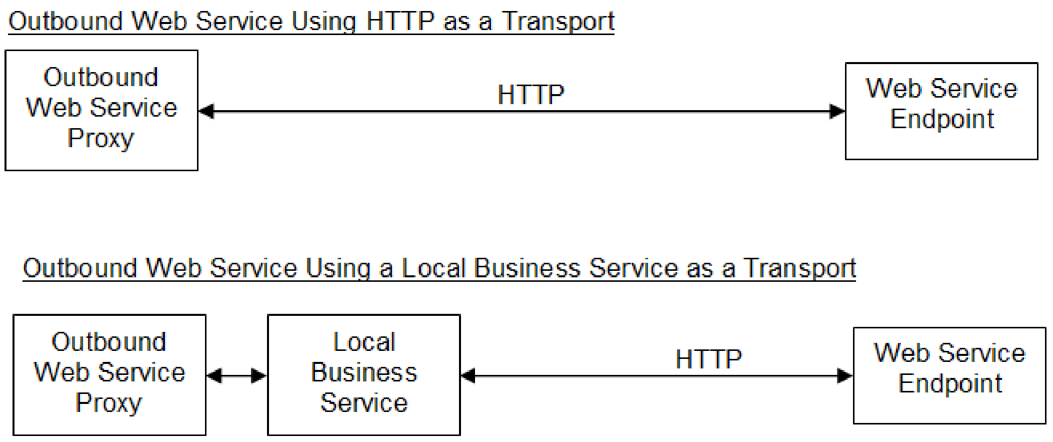

Outbound Web Service Using HTTP as a Transport
Outbound Web Service Proxy
HTTP
Web Service Endpoint
Outbound Web Service Using a Local Business Service as a Transport
Outbound Web Service Proxy
Local Business Service
HTTP
Web Service Endpoint

Figure 25. Local Business Service Used as a Transport

If the Web Service to be invoked is within the sample application, then no need exists to go through HTTP (or anything else) to invoke such a Web Service.

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. The following code sample shows an example where a local business service was used to add a custom SOAP header to an outbound Web Service request.

```
function Service_PreInvokeMethod (MethodName, Inputs, Outputs)
{
            // local variables & error handling  omitted for clarity
            soapHdr.SetType("SOAP-ENV:header");

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

            // populate the eProfileHeader element
            profileHeader.SetType("authHeader");
            profileHeader.SetProperty("xmlns:ns1", "http://siebel.com/authHeaders");
            profileHeader.AddChild(appId);
```

```
18          profileHeader.AddChild(pwd);
19          profileHeader.AddChild(langCd);
20          profileHeader.AddChild(uName);
21
22          // SOAP header property set.  Once this is complete, add the SOAP header
23          // as a child of the Input property set (which contains the SOAP:body
24          soapHdr.InsertChildAt(profileHeader, 0)
25          Inputs.InsertChildAt(soapHdr, 0);
26
27          // convert property set to well defined SOAP/XML document
28          // due to XML Hierarchy Converter, need to create add a child element of type XMLHierarchy
29          childPS.SetType("XMLHierarchy");
30          childPS.AddChild(Inputs);
31          inPs.AddChild(childPS);
32          inPs.SetProperty("EscapeNames", "FALSE");
33          inPs.SetProperty("GenerateProcessingInstructions", "FALSE");
34          xmlSvc.InvokeMethod("XMLHierToXMLDoc", inPs, outPs);
35
36          // proxy the request through trace utility to view SOAP document
37          // set custom HTTP header - SOAPAction
38          outPs.SetProperty("HTTPRequestURLTemplate", "http://localhost:9000/search/beta2");
39          outPs.SetProperty("HTTPRequestMethod", "POST");
40          outPs.SetProperty("HTTPContentType", "text/xml; charset=UTF-8");
41          outPs.SetProperty("HDR.SOAPAction","customSOAPActionValue");
42
43          // invoke Web Service using standard HTTP protocol
44          httpSvc.InvokeMethod("SendReceive", outPs, hpOut);
45
46          // Converting the SOAP document to a XMLHierarchy propset
47          xmlSvc.InvokeMethod("XMLDocToXMLHier", hpOut, tmp);
48
49          // removing XMLHierarchy, returning the SOAP header and SOAP body
50          soapDoc = tmp.GetChild(0).GetChild(0);
51          Outputs.AddChild(soapDoc);
52
53          return (CancelOperation);
```

The following XML code sample displays the resulting SOAP document generated by the local business service. Note the addition of the <authHeader> element in the SOAP header which corresponds to the structure defined between lines 4 - 20 in the preceding code sample.

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

        ...

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

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

In the current framework, WSDL Import Wizard makes use of XML Schema Import Wizard to create integration objects to represent hierarchical data. Integration objects are meant to be strongly typed in the Siebel application. You are now able to import a schema that uses the <xsd:any> tag, which indicates a weakly typed data representation, and to 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 not known, then the business service method argument for that particular <wsdl:part> tag will be changed to data type Hierarchy, consequently losing any type information.

Being known refers to the following situations:

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

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

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

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

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

## 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 not, an error will be returned to the user saying that the integration object cannot be created for a weakly typed schema.

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

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

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

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

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

## Examples of Invoking Web Services

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

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

As illustrated on Figure 26 on page 89, 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.

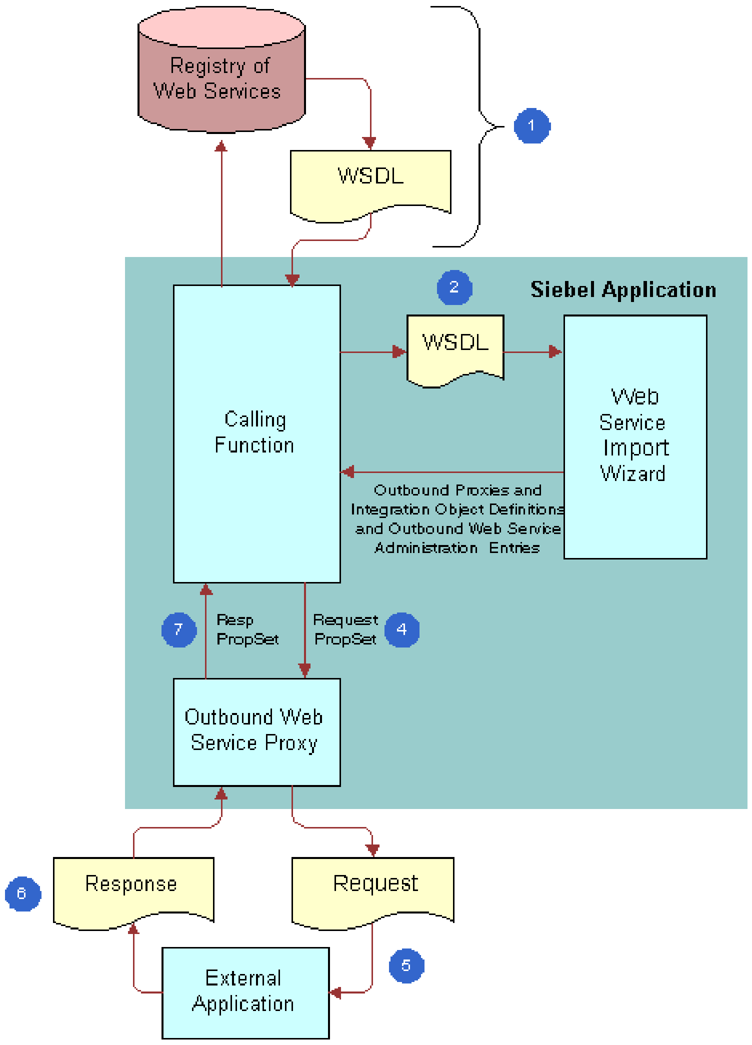

Registry of Web Services
WSDL
1
Siebel Application
2
WSDL
Calling Function
Web Service Import Wizard
Outbound Proxies and Integration Object Definitions and Outbound Web Service Administration Entries
7
Resp PropSet
Request PropSet
4
Outbound Web Service Proxy
6
Response
Request
External Application
5

Figure 26. 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 27 on page 91, the following steps are executed to invoke a Siebel Web Service from an external application.

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

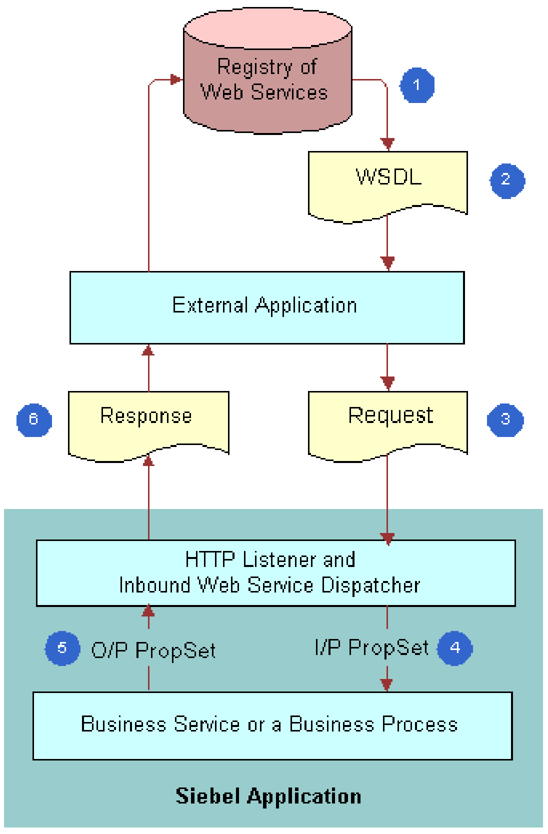


Figure 27. Invoking a Siebel Web Service

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 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. Release 7.7 introduces support for the UserName Token mechanism of the WS-Security specification, which allows Siebel applications to send and receive user credentials in a standards-compliant manner.

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.

### 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 in the eapps.cfg file.

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

- In the eapps.cfg file, make sure that these entries in the Secure Web Services named subsystem read as follows:

  ```
  UseAnonPool = TRUE
  ```

  ```
  Impersonate = TRUE
  ```

## About the WS-Security UserName Token Profile Support

Release 7.7 introduces support for WS-Security's UserName Token mechanism, allowing Siebel applications to send and receive 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.

Following is a sample of the UserName token showing the username and password.

```
<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>
```

### About Support for the UserName Token Mechanism

The 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.

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
```

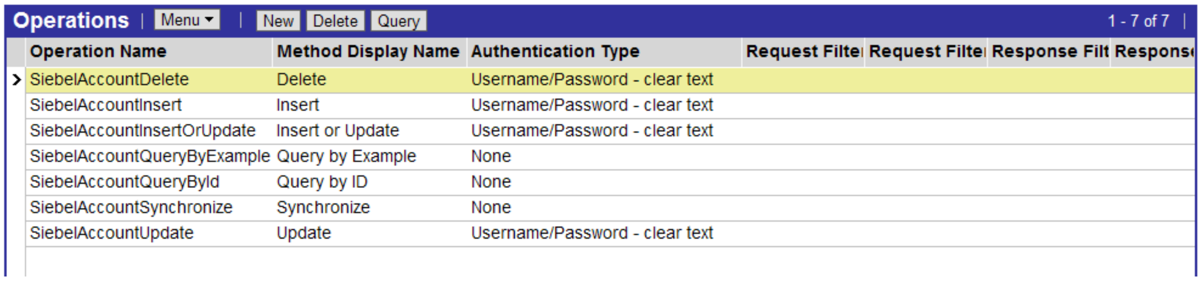
With UserName tokens, the URL does not reveal user credentials:

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

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

## About Using the UserName Token for Inbound Web Services

The Inbound Web Services Administration screen provides an interface for associating operations with authentication types. The names of the operations need to be globally unique. The applet shown in Figure 28 can be defined as requiring no authentication or requiring a UserName Token with username and password provided in clear text.



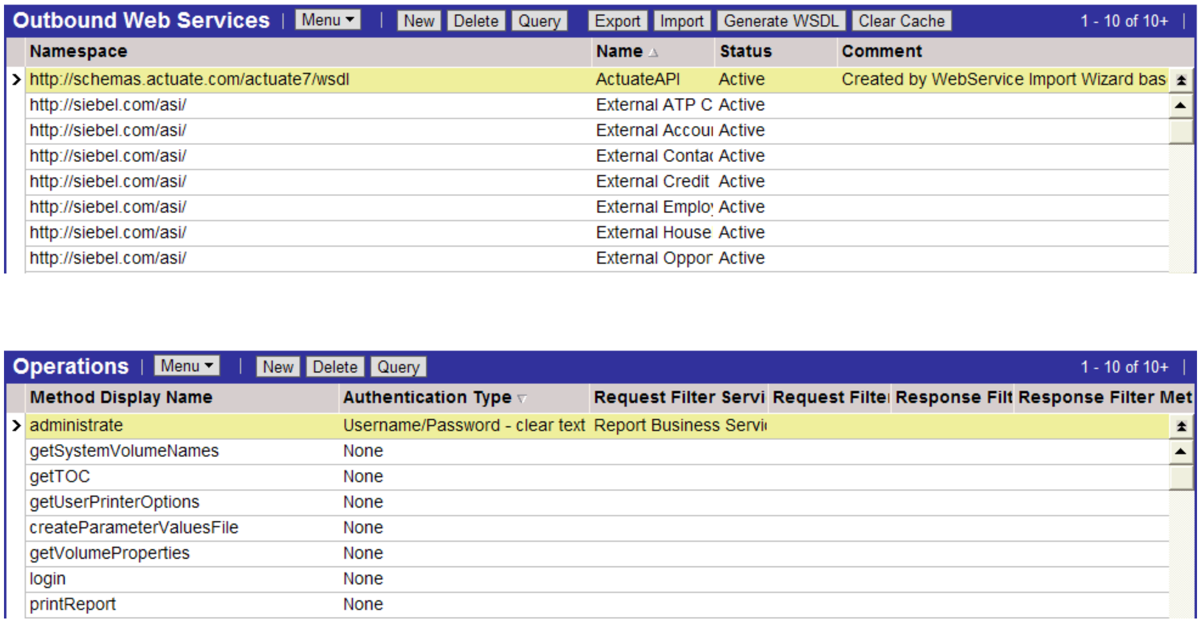
| Operation Name | Method Display Name | Authentication Type | Request Filte | Request Filte | Response Filt | Respons |
|---|---|---|---|---|---|---|
| SiebelAccountDelete | Delete | Username/Password - clear text | | | | |
| SiebelAccountInsert | Insert | Username/Password - clear text | | | | |
| SiebelAccountInsertOrUpdate | Insert or Update | Username/Password - clear text | | | | |
| SiebelAccountQueryByExample | Query by Example | None | | | | |
| SiebelAccountQueryById | Query by ID | None | | | | |
| SiebelAccountSynchronize | Synchronize | None | | | | |
| SiebelAccountUpdate | Update | Username/Password - clear text | | | | |

Figure 28. Inbound Web Services Administration Screen 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 29) in the Operations picklist, in the applet shown below.



| Namespace | Name | Status | Comment |
|---|---|---|---|
| http://schemas.actuate.com/actuate7/wsdl | ActuateAPI | Active | Created by WebService Import Wizard bas |
| http://siebel.com/asi/ | External ATP C | Active | |
| http://siebel.com/asi/ | External Accou | Active | |
| http://siebel.com/asi/ | External Conta | Active | |
| http://siebel.com/asi/ | External Credit | Active | |
| http://siebel.com/asi/ | External Emplo | Active | |
| http://siebel.com/asi/ | External House | Active | |
| http://siebel.com/asi/ | External Oppor | Active | |

| Method Display Name | Authentication Type | Request Filter Servi | Request Filte | Response Filt | Response Filter Met |
|---|---|---|---|---|---|
| administrate | Username/Password - clear text | Report Business Servi | | | |
| getSystemVolumeNames | None | | | | |
| getTOC | None | | | | |
| getUserPrinterOptions | None | | | | |
| createParameterValuesFile | None | | | | |
| getVolumeProperties | None | | | | |
| login | None | | | | |
| printReport | None | | | | |

Figure 29. Outbound Web Services Administration Screen and the Operations PickList

# 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' = 1, the custom header is interpreted as being mandatory and the custom header is processed by the filter defined for this purpose.

- If 'mustUnderstand' = 1 and a filter is not specified, the custom header is not read and a SOAP:MustUnderstand fault is generated.
- If 'mustUnderstand' = 0, no processing of the custom header is attempted.

You want to keep SOAP body and header processing isolated. The inbound dispatcher and outbound proxy know how to process the SOAP body but have no idea on 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. The Web Service outbound proxy and the Web Service inbound dispatcher can be configured 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. The methods to be invoked on the filter can also be defined.

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 would like 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 these headers information out. 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 will pass *HeaderContext* to the eventual business service or workflow that is invoked.

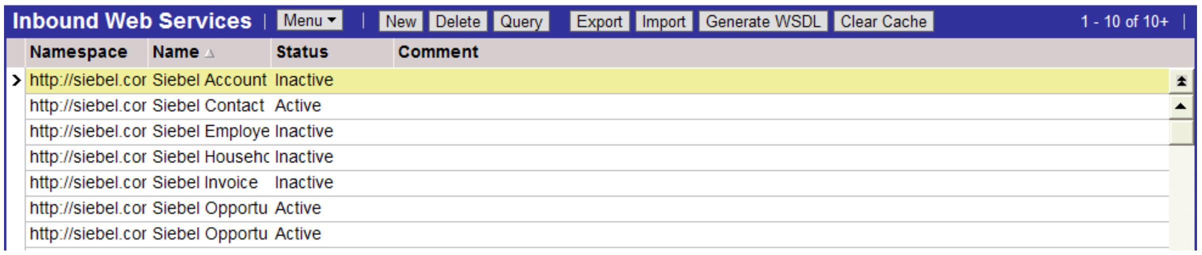
## About Web Services Cache Refresh

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 the need to stop and restart Siebel services.

Web Services cache is used to store all the global administration information that can be manipulated in the Inbound and Outbound Web Service administration screens.

The Clear Cache feature requires user interaction. The administrator decides when Web Service configuration needs to 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 in the Administration - Web Services screen. This feature is available for inbound and outbound Web Services. Figure 30 reflects this feature for inbound Web Services.



| Namespace | Name | Status | Comment |
|---|---|---|---|
| http://siebel.cor | Siebel Account | Inactive | |
| http://siebel.cor | Siebel Contact | Active | |
| http://siebel.cor | Siebel Employe | Inactive | |
| http://siebel.cor | Siebel Househc | Inactive | |
| http://siebel.cor | Siebel Invoice | Inactive | |
| http://siebel.cor | Siebel Opportu | Active | |
| http://siebel.cor | Siebel Opportu | Active | |

Figure 30. Clear Cache Button

## 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 From the application-level menu, choose Navigate > Site Map > Administration - Server Configuration > Servers.

   The screen that appears displays three different list applets. The top applet lists the Siebel Servers for the enterprise. The middle applet has three tabs—Components, Parameters and Events. The bottom applet has two tabs—Events and Parameters.

2 In the top list applet, select the Siebel Server that you want to configure.

3 In the middle applet, click the Components tab.

   This list applet contains the components for the Siebel Server selected in the top applet.

   Choose the relevant application object manager.

4 In the bottom applet, click the Parameters tab.

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

5 Set the Log Level to 4 for any or all of the following Event Types:

| 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. |

6 Navigate to the Components view.

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

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

9 Restart or reconfigure the server component. For details, see the *Siebel System Administration Guide*.

## About Integration Components Cardinality

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

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

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

# 6 EAI Siebel Adapter

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.

This chapter includes the following topics:

## 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 whose data originates externally into a Siebel business object.
- Update multiple corresponding top-level parent business component records with data from one XML file—for examples, see "XML Examples Using the Upsert and Delete Operation" on page 110.

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

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

## EAI Siebel Adapter Methods

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

- Query
- QueryPage
- Synchronize

- Upsert
- Insert
- Update
- Delete
- Execute

**NOTE:** 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 should update these types of data.

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

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

## EAI Siebel Adapter Method Arguments

Each of the EAI Siebel Adapter methods takes arguments that allow you to specify required and optional information to the adapter. You can locate the arguments for each method in Table 9.

Table 9. EAI Siebel Adapter Method Arguments

| Argument | Query | QueryPage | Sync | Upsert | Update | Insert | Delete | Execute |
|---|---|---|---|---|---|---|---|---|
| IntObjectName | - | - | - | - | - | ‘ | ‘ ‘ | Input |
| NumOutputObjects | Output | Output | Output | Output | Output | Output | Output | Output |
| OutputIntObjectName | Input | Input | - | - | - | - | - | Input |
| PrimaryRowId | Input | - | Output | Output | Output | Output | Input | Input/ Output |
| QueryByUserKey | Input | - | - | - | - | - | - | Input |
| DeleteByUserKey | - | - | - | - | - | - | Input | Input |
| ErrorOnNonExistingDelete | - | - | - | - | - | - | Input | Input |

Table 9. EAI Siebel Adapter Method Arguments

| Argument | Query | QueryPage | Sync | Upsert | Update | Insert | Delete | Execute |
|---|---|---|---|---|---|---|---|---|
| SiebelMessage | Input/ Output | Input/ Output | Input/ Output | Input/ Output | Input/ Output | Input/ Output | Input/ Output | Input/ Output |
| SearchSpec | Input | Input | - | - | - | - | Input | Input |
| StatusObject | - | - | Input | Input | Input | Input | Input | Input |
| MessageId | Input | Input | Input | Input | Input | Input | Input | Input |
| BusObjCacheSize | Input | Input | Input | Input | Input | Input | Input | Input |
| LastPage | - | Output | - | - | - | - | - | Output |
| NewQuery | - | Input | - | - | - | - | - | Input |
| PageSize | - | Input | - | - | - | - | - | Input |
| StartRowNum | - | Input | - | - | - | - | - | Input |
| ViewMode | Input | Input | Input | Input | Input | Input | Input | Input |
| SortSpec | - | Input | - | - | - | - | - | Input |

Table 10 presents each argument of EAI Siebel Adapter methods.

Table 10. EAI Siebel Adapter Method Arguments

| Argument | Display Name | Description |
|---|---|---|
| IntObjectName | Integration Object Name | The name of the integration object that is to be deleted. |
| NumOutputObjects | Number of Output Integration Objects | Number of output integration objects. |
| OutputIntObjectName | Output Integration Object Name | The name of the integration object that is to be output. |
| PrimaryRowId | Object Id | The PrimaryRowId refers to the Id field in the Business Component, Row_Id at the table level.<br><br>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. |

Table 10. EAI Siebel Adapter Method Arguments

| Argument | Display Name | Description |
|---|---|---|
| QueryByUserKey | Query By Key | A Boolean argument. Forces the EAI Siebel Adapter to only use the user keys to perform query. |
| DeleteByUserKey | Delete By User Key | A Boolean argument. Forces the EAI Siebel Adapter to only use the user keys to identify a record. |
| ErrorOnNonExistingDelete | Error On Non Existing Delete | A Boolean argument. Determines whether or not the EAI Siebel Adapter should abort the operation if no match is found. |
| SiebelMessage | Siebel Message | The input or the output integration object instance. |
| SearchSpec | Search Specification | This argument allows you to specify complex search specifications as free text in a single method argument. See "About the SearchSpec Input Method Argument" on page 112 for details. |
| StatusObject | Status Object | This argument tells EAI Siebel Adapter whether or not to return a status message. |
| MessageId | Message Id | The MessageId can be used to specify the ID for the generated message. By default, the EAI Siebel Adapter generates a unique ID for each message. However, if you want to use the workflow process instance ID, then you can use this argument to specify the ID. |
| BusObjCacheSize | Business Object Cache Size | Default is 5. Maximum number of Business Objects instances cached by the current instance of the EAI Siebel Adapter. If set to zero, then the EAI Siebel Adapter does not use the cache. |
| LastPage | Last Page | Boolean indicating whether or not the last record in the query result set has been returned. |
| NewQuery | New Query | Default is False. Boolean indicating whether a new query should be executed. If set to True, a new query is executed flushing the cache for that particular integration object. |
| PageSize | Page Size | Default is 10. Indicates the maximum number of integration object instances to be returned. |
| StartRowNum | Starting Row Number | Default is 0 (first page). Indicates the row in the result set for the QueryPage method to start retrieving a page of records. |

Table 10. EAI Siebel Adapter Method Arguments

| Argument | Display Name | Description |
|---|---|---|
| ViewMode | View Mode | Default is All. Visibility mode to be applied to the Business Object. Valid values are: Manager, Sales Rep, Personal, Organization, Sub-Organization, Group, Catalog, and All. Note that the ViewMode user property on the integration object has priority over the ViewMode method argument. |
| SortSpec\ | Sort Specification | Default is the SortSpec of the underlying business component. This argument allows you to specify complex sort criteria as a free text in a single method argument, using any business component fields and standard Siebel sort syntax—for examples, see *Using Siebel Tools*. |

## About the Query Method

You pass the Query method using only one of the following method arguments:

- A Query By Example (QBE) integration object instance.
- A Primary Row Id.
- A Search Specification.

The adapter uses this input as criteria to query the base business object and to return a corresponding integration object instance. For example, to query Contact records with first name *David* you need to pass the following required input arguments to the Query method of EAI Siebel Adapter:

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

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

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

Note that this still returns the contacts with the first name *David*, even if they do not have an activity of type *Call*, but it does not list their activities. For an example of using the search specification method argument to limit the scope of your query see "About the SearchSpec Input Method Argument" on page 112.

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

## About the 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 10 on page 103. 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.

## About the Synchronize Method

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

- If a child component is not present in the integration object instance, the corresponding business component rows are left untouched.
- If a child component is present in the integration object instance, but contains no instances so that there is only an empty container, then records in the corresponding business component are deleted.
- If a child component is present in the integration object instance, and contains some instances, the business component rows corresponding to the instances are updated or created and any business component row that does not have a corresponding integration component instance is deleted.

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

## About the Upsert Method

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

## About the Insert Method

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

## About the Update Method

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

1. Queries for the order and if it finds a match, it updates the record.
2. Updates or inserts the new order item depending on if a match was found for the new order item.
3. Deletes any other order items associated with that order.

## About the Delete Method

You can delete one or more records in a business component that is mapped to the root integration component, given an integration object. A business component is deleted as specified by an integration object. If you specify any child integration component instances, then the fields of an integration component instance are used to query a business component.

**NOTE:** To have the EAI Siebel Adapter perform a delete operation, define an integration object that contains the minimum fields on the primary business component for the business object. EAI Siebel Adapter attempts to delete matching records in the business component before deleting the parent record.

## About the Execute Method

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

- query
- querypage (same as query when used as children operation)
- sync (default operation)
- upsert
- update
- updatesync
- insert
- insertsync
- delete
- none

**NOTE:** A none operation is equivalent to operation sync.

These operations perform the same tasks as the related methods. For example, the delete operation makes the EAI Siebel Adapter delete the business component record matched to the particular integration component instance. However, what will be done to the children depends on the combination of the parent operation and the child operation. For details, see Table 12 on page 109.

Operations that include the word sync in the name cause deletion of unmatched child records, whereas update, insert, and upsert do not delete any children. Table 11 presents the overview of the six related operations.

Table 11. EAI Siebel Adapter Execute Method Operations

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

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

An XML document sent to a Siebel application can include operations that describe whether a particular data element needs to be inserted, updated, deleted, synchronized, and so on. These operations can be specified as an attribute at the component level. They cannot be specified for any other element.

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

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

### Supported Operations for the Parent and Its Child Components

Table 12 presents the operation performed for a child component based on its parent component's operation and its own operation.

Table 12. Supported Operations

| | | **Parent Operation** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | **query** | **query page** | **sync** | **upsert** | **update** | **update sync** | **insert** | **insert sync** | **delete** |
| **Child Operation** | **query** | query | query | upsert | upsert | update | update | insert | insert | delete |
| | **query page** | query | query | upsert | upsert | update | update | insert | insert | delete |
| | **sync** | query | query | sync | sync | sync | sync | sync | sync | delete |
| | **upsert** | query | query | upsert | upsert | upsert | upsert | upsert | upsert | delete |
| | **update** | query | query | update | update | update | update | upsert | upsert | delete |
| | **update sync** | query | query | update sync | update sync | update sync | update sync | sync | sync | delete |
| | **insert** | query | query | insert | insert | insert | insert | insert | insert | delete |
| | **insert sync** | query | query | insert sync | insert sync | insert sync | insert sync | insert sync | insert sync | delete |
| | **delete** | query | query | delete | delete | delete | delete | delete | delete | delete |

## XML Examples Using the Upsert and Delete Operation

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

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

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

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

# About MVGs in EAI Siebel Adapter

Multi-value groups (MVGs) in the business components are mapped to separate integration components. Such integration components are denoted by setting a user property *MVG* on the integration component to *Y*. For details on MVGs, see Chapter 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 details on Auto Primary property, see *Siebel Tools Online Help*.

## Setting a Primary Address for an Account

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

***To specify an address as a primary***

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

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

```
<EmployeeFirstName>Siebel</EmployeeFirstName>
<EmployeeLastName>Administrator</EmployeeLastName>
<Position>Siebel Administrator</Position>
<RowStatus>N</RowStatus>
<SalesRep>SADMIN</SalesRep>
</Contact_Position>
</ListOfContact_Position>
</Contact>
</ListOfSampleContact>
</SiebelMessage>.
```

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

## About the SearchSpec Input Method Argument

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

[*IntCompName.IntCompFieldName*]

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

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

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

**NOTE:** The AND operator is the only allowed operator among different integration components. You use DOT notation to refer to integration components and their fields.

You can include the child integration component in a search specification only if its parent components are also included. For example, using the same integration object definition as in previous examples, the `[Contact.PhoneNumber] IS NOT NULL` queries every account. Then for each account, it queries only contacts that have a phone number. If you want to query only accounts that are associated with contacts that have a phone number specified, then you need to create another business object, and an integration object based on that business object, which has contact as a root component, and account as its child component.

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

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

1 From the application-level menu, choose Navigate > Site Map > Administration - Business Process > Workflow Processes.

**2** Create a new workflow process based on the Sample Account business object.

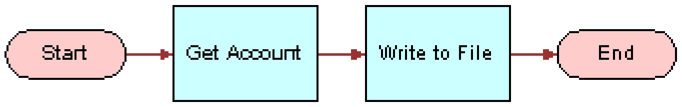
**NOTE:** Make sure all the fields you need are activated in the object.

**3** Define the process properties.

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

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

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



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

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

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

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

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

```
<?xml version="1.0" encoding="UTF-8" ?>
   <?Siebel-Property-Set EscapeNames="false"?>
- <SiebelMessage MessageId="1IS-7LT" IntObjectFormat="Siebel Hierarchical"
MessageType="Integration Object" IntObjectName="Sample Account">
- <ListOfSampleAccount>
- <Account>
   <Created>04/05/2002 07:41:10</Created>
   <CSN>1IS-1DBRT</CSN>
   <Location>Princeton</Location>
   <Name>1st Account created today</Name>
- <ListOfAccount_BusinessAddress>
- <Account_BusinessAddress IsPrimaryMVG="N">
   <City>Abbott Park</City>
   <Country>USA</Country>
   <State>IL</State>
   <StreetAddress>1 Abbott Rd. D3m, B 3</StreetAddress>
   <AddressName>1 Abbott Rd. D3m, B 3, Abbott Park, IL</AddressName>
   </Account_BusinessAddress>
   </ListOfAccount_BusinessAddress>
   </Account>
- <Account>
   <Created>04/05/2002 07:42:27</Created>
   <CSN>1IS-1DBRY</CSN>
   <Location>Orange</Location>
   <Name>2nd Account created today</Name>
- <ListOfAccount_BusinessAddress>
- <Account_BusinessAddress IsPrimaryMVG="Y">
   <City>Chicago</City>
   <Country>USA</Country>
   <State>IL</State>
   <StreetAddress>1 BOP, 7th Floor</StreetAddress>
   <AddressName>1 BOP, 7th Floor, Chicago, IL</AddressName>
   </Account_BusinessAddress>
   </ListOfAccount_BusinessAddress>
   </Account>
   </ListOfSampleAccount>
   </SiebelMessage>
```

## About Using Language-Independent Code with the Siebel Adapter

If the user Property AllLangIndependentVals is set to Y at the integration object level, then EAI Siebel Adapter uses the language-independent code for its LOVs.

In the outbound direction, for example the Query method, if the AllLangIndependentVals is set to Y, then the EAI Siebel Adapter translates the language-dependent values in the Siebel Database to their language-independent counterpart based on the List Of Values entries in the database.

In the inbound direction, for example the Synchronize method, if the AllLangIndependentVals is set to Y, then the EAI Siebel Adapter expects language-independent values in the input message, and translates them to language-dependent values based on the current language setting and the entries in the List Of Values in the database.

**NOTE:** The LOV-based fields are always validated using language-dependent values. Using language independent values for (M)LOVs increases the EAI Siebel Adapter CPU usage by about 5%, but allows easier communication between systems that operate on different languages.

### 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 gets translated to a language-dependent value (LDV) for active language by Object Manager. MLOVs are distinguished by having Translation Table specified on the Column definition.

Single-language LOV stores the LDV for the current language in the Siebel Database. The Boolean integration object user property *AllLangIndependentVals* determines whether the EAI Siebel Adapter should use LDV (N = no translation necessary) or LIC (Y = translation needed) for such LOVs.

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

Table 13 explains the behavior of the EAI Siebel Adapter according to the integration object user property *AllLangIndependentVals* values.

Table 13. EAI Siebel Adapter's Behavior for the User Property AllLangIndependentVals

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

# 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 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.

EAI Siebel Adapter methods (Insert, Update, Synchronize, Upsert) check for the existence of a Modification Key. If no Modification Key is specified in the integration component definition, or if Modification Key fields are not included in the XML request, the EAI Siebel Adapter does not check for the record version and proceeds with the requested operation. If a valid Modification Key is found, but the corresponding record can not be found, the EAI Siebel Adapter assumes that the record has been deleted by other users and returns the error SSASqlErrWriteConflict.

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

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

### 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 need to 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 need to 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 needs to be manually added in both business component and integration object definitions. Business component Mod Id fields for joined tables should:

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

## About MVG and MVGAssociation Integration Components

For integration components that are of type MVG or MVGAssociation, in addition to the above steps, you need to create user properties MVGFieldName and AssocFieldName for each Modification ID integration component field, respectively, and set the name of the field shown in the parent business component as the value.

***To configure EAI Siebel Adapter for concurrency control***

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

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

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

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

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

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

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

2. Create business component fields for Mod Ids of Joined Tables. For the above example, create a new field in business component Account with the following settings:

   **Name.** CX_Primary Organization-S_BU.Mod Id

   **Join.** Primary Organization-S_BU

   **Column.** MODIFICATION_NUM

   **Type.** DTYPE_INTEGER

3. Expose all Modification IDs identified in Step 1 as integration component fields.
4. For MVG and MVG Association integration components, add user property MVGFieldName and AssocFieldName respectively, on all Modification ID fields as follows:
   a. Check the Integration Component User Prop sub type for user properties of the integration component.
   b. If there is a user property called MVGAssociation then the integration component is a MVG Association, but if there is a user property called Association then the integration component is a MVG.

      **NOTE:** If the integration component is neither an MVG nor an MVG Association, then nothing needs to be done.
5. Repeat the following steps for each Modification ID field on the integration component.
   a. Add user property MVGFieldName if MVG, or AssocFieldName if MVG Association.
   b. Set the value of the user property to the same as the field name—for example, Mod Id, *extension table name*.Mod Id, or CX_*join*.Mod Id.
6. Create Modification Key.

   Define a new integration component key of type Modification Key, and include all the integration component fields exposed in Step 3 to this key.
7. Validate integration objects and compile a new SRF.
8. Modify client program to use the Modification Key mechanism.
   a. The client program should store the value of the Modification IDs when it queries data from Siebel Database.
   b. The client program should send exactly the same values of the Modification IDs that it retrieved from Siebel Database when sending an update.
   c. The client program should not send in any Modification IDs when sending a new record to the Siebel application. If this is violated, the client program generates an error indicating that the record has been deleted by another user.

### Integration Component Account Example

Consider an integration component Account of the business component Account:

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

- Field Primary Organization has property Join set to Primary Organization-S_BU. This is an explicit join as it is listed in the joins. The value of Table property is S_BU; therefore, this field belongs to Joined Table S_BU joined over Primary Organization-S_BU.

1. Activate integration component field Mod Id.
   a. Set Name, External Name, XML Tag properties to Mod Id
   b. Set External Data Type property to DTYPE_NUMBER
   c. Set External Length property to 30
   d. Set Type property to System
2. Add integration component field S_ORG_EXT.Mod Id.
   a. Set Name, External Name, XML Tag properties to S_ORG_EXT.Mod Id
   b. Set External Data Type property to DTYPE_NUMBER
   c. Set External Length property to 30
   d. Set Type property to System
3. Add integration component field CX_Primary Organization-S_BU.Mod Id.
   a. Set Name, External Name, XML Tag properties to CX_Primary Organization-S_BU.Mod Id
   b. Set External Data Type property to DTYPE_NUMBER
   c. Set External Length property to 30
   d. Set Type property to System

### Integration Component Account_Organization Example

Consider the integration component Account_Organization of the Sample Account integration object. Account_Organization is an MVG Association as denoted by the presence of the user property MVGAssociation. Assume two Modification IDs, Mod Id and S_ORG_EXT.Mod Id, were exposed on this integration component.

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

In the integration component example, Account (created in "Integration Component Account_Organization Example" on page 119) of Sample Account integration object, takes the following action:

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

## Siebel eAI and Run-Time Events

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

**Run-Time Events.** Siebel eAI supports triggering workflows based on run-time events such as Write Record, which gets triggered whenever a record is written. If you use both EAI Siebel Adapter to import data into Siebel application and run-time events, you should pay attention to the following:

For EAI Siebel Adapter, one call to EAI Siebel Adapter with an input message is a transaction. Within a transaction, EAI Siebel Adapter makes multiple Write Record calls. At any point in the transaction, if EAI Siebel Adapter encounters a problem the transaction is rolled back entirely. However, if you have specified events to trigger at Write Record, such events are invoked as soon as EAI Siebel Adapter makes Write Record calls even though EAI Siebel Adapter may be in the middle of a transaction. If you have export data workflows triggered on such events, this may lead to exporting data from Siebel applications that is not committed in Siebel applications and may get rolled back. It is also possible that your events get triggered when the record is not completely populated, which leads to situations that are not handled by your specified event processing workflow.

To avoid the effects of this interaction between EAI Siebel Adapter and run-time events use the business service EAI Transaction Service to figure out if a transaction (typically, EAI Siebel Adapter) is in progress. You may then want to skip processing that is not desirable when EAI Siebel Adapter is in progress.

For example, suppose you have a workflow to export Orders from Siebel applications that is triggered whenever the Order record is written. You also import Orders into Siebel applications using EAI. In such a situation, you do not want to export Orders while they are being imported because the import may get aborted and rolled back. You achieve this using the business service EAI Transaction Service as the first step of the export workflow. If you find that a transaction is in process you can branch directly to the end step.

**Workflow Policies.** In addition to Run-Time Events, Siebel applications also support Workflow Policies as a triggering mechanism for workflows. You can use workflow policies instead of run-time events to avoid the situation discussed above. You should use Workflow Policies instead of Run-Time Events when possible.

# 7 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 details, 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 multi-part messages), or including the attachment within the body of the XML document, referred to as an inline XML attachment. You should consider using inline XML attachments when integrating two instances of Siebel applications using file attachments.

## Exchange of Attachments with External Applications

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

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

- **Inline XML attachments (Inline Extensible Markup Language).** With inline XML attachments, the entire business object you are exchanging, including any attachments, is sent as a single XML file. In this case, attachments are included within the body of the inline XML attachment. Inline XML attachments should be considered when integrating two instances of Siebel applications using file attachments. For details, see *XML Reference: Siebel eBusiness Application Integration Volume V*.

## Using MIME Messages to Exchange Attachments

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

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

| Checklist | |
|---|---|
| ❑ | Create an integration object using the EAI Siebel Wizard.<br><br>For details, see "Creating an Integration Object" on page 122. |
| ❑ | Create an inbound or outbound Workflow process.<br><br>For details, see "Creating Workflow Processes Examples" on page 123. |
| ❑ | Test your workflow process using Workflow Process Simulator.<br><br>For details, see "About the EAI MIME Hierarchy Converter" on page 128. |

### Creating an Integration Object

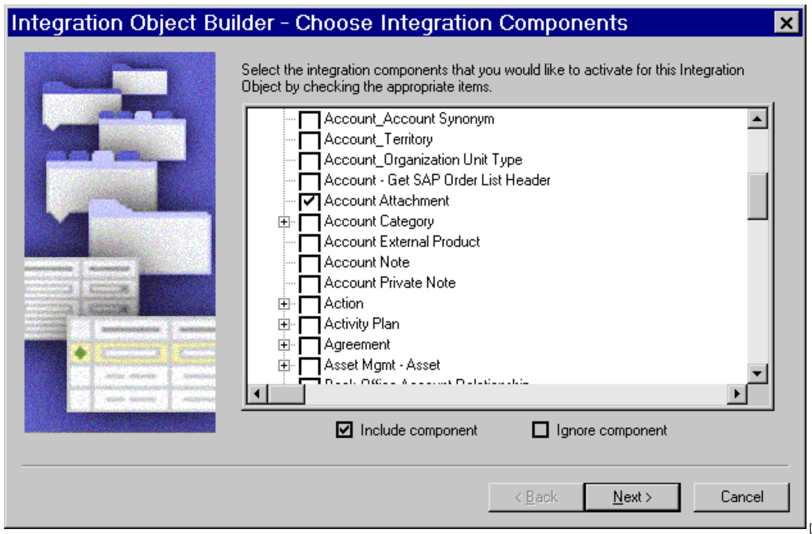
The following procedure guides you through the steps of creating an integration object.

#### *To create a new Siebel integration object*

1. Start Siebel Tools.
2. Create a new project and lock the project, or lock an existing project in which you want to create your integration object.
3. Choose File > New Object... to display the New Object Wizards dialog box.

4 Select the EAI tab, double-click Integration Object, and click OK.

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


Integration Object Builder - Choose Integration Components

Select the integration components that you would like to activate for this Integration Object by checking the appropriate items.

- [ ] Account_Account Synonym
- [ ] Account_Territory
- [ ] Account_Organization Unit Type
- [ ] Account - Get SAP Order List Header
- [x] Account Attachment
- [ ] Account Category
- [ ] Account External Product
- [ ] Account Note
- [ ] Account Private Note
- [ ] Action
- [ ] Activity Plan
- [ ] Agreement
- [ ] Asset Mgmt - Asset

[✓] Include component   [ ] Ignore component

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

6 Review and take necessary actions to address the issue.

7 Click Finish to complete the process of building the integration object.

8 In the Object Explorer, select Integration Object > Integration Component > Integration Component Field object.

The Integration Component and Integration Component Field applets appear.

9 Select the XXX_Attachment Component and the Attachment Id Component fields, and verify that the Data Type object for the Attachment Id field is set to DTYPE_ATTACHMENT.

10 Compile the SRF file and copy it to the object directory under your Siebel Server directory as well as under your Tools directory.

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

## Creating Workflow Processes Examples

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

## Outbound Workflow Process

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

### *To create an outbound workflow process*

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

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

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

| Property Name | Type | Output Argument |
|---|---|---|
| SiebelMessage | Output Argument | Siebel Message |

**NOTE:** For more information on using EAI Siebel Adapter, see Chapter 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 128.

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 129.

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

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

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

## Inbound Workflow Process Example

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

### *To create an inbound workflow process*

1 Navigate to Workflow Process Designer.

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

3 Define your process properties.

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

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

5 The second business service of the workflow converts the Account information to a MIME hierarchy using the EAI MIME Doc Converter business service with the MIME Doc to MIME Hierarchy method. This step requires the following input and output arguments.

| Input Argument | Type | Property Name | Property Data Type |
|---|---|---|---|
| MIME Message | Process Property | <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 31 illustrates the Siebel Message of a sample Account with attachments. This figure represents both input and output to the MIME Hierarchy Converter.

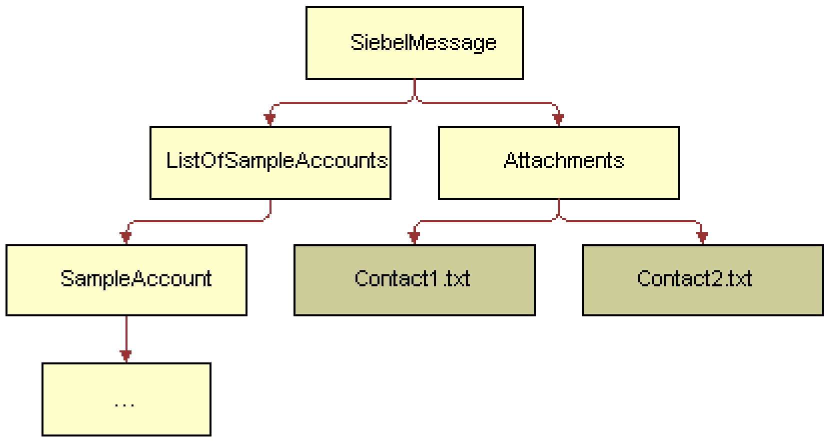


Figure 31. Sample Account with Attachments as Input to the MIME Hierarchy Converter

The output of this process is illustrated in Figure 32.

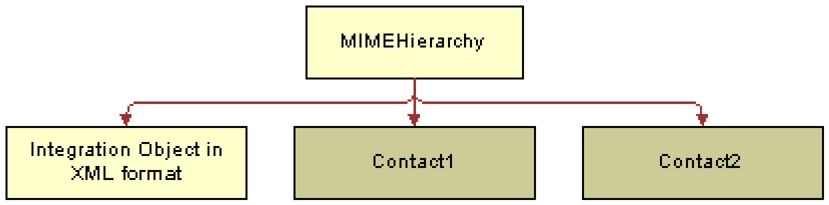


Figure 32. Output of a MIME Hierarchy Converter

The first child of a MIME Hierarchy is the XML format of the Sample Account Integration Object instance found in the Siebel Message. The remaining two children are the corresponding children found under Attachments. In the event that there is no child of type Attachments in the Siebel Message, the output is just a MIME Hierarchy with a child of type Document. This document will contain the XML format of the Sample Account integration object instance.

## Inbound Integration

The MIME Hierarchy Converter transforms a MIME Hierarchy input into a Siebel Message. For the inbound process, the first child of the MIME Hierarchy has to be the XML format of the Integration Object instance; otherwise, an error is generated. Figure 33 illustrates the incoming hierarchy.

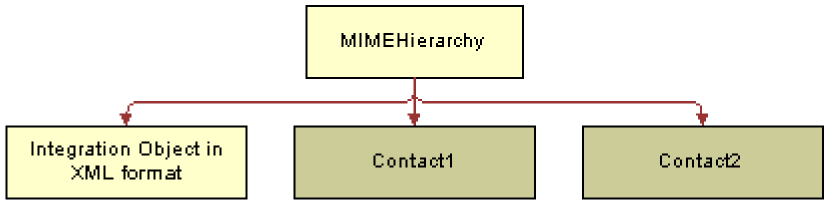


Figure 33. Output of a MIME Hierarchy Converter

The output of this process is illustrated in Figure 31 on page 128. The output for this process is the same as the input.

## About the EAI MIME Doc Converter

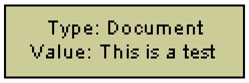
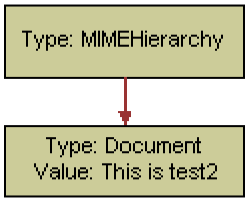
The MIME Doc Converter converts a MIME Hierarchy into a MIME Message and a MIME Message into a MIME Hierarchy. A MIME Hierarchy consists of two different types of property sets.

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

Table 14 illustrates some examples of how a MIME Message maps to a MIME Hierarchy.

## EAI MIME Doc Converter Properties

Table 14. Examples of MIME Message and MIME Hierarchy

| MIME Message | MIME Hierarchy |
|---|---|
| MIME-Version: 1.0<br><br>Content-Type: application/xml<br><br>Content-Transfer-Encoding: 7bit<br><br>This is a test. |  |
| MIME-Version: 1.0<br><br>Content-Type: multipart/related; type="application/xml"; boundary=--abc<br><br>----abc<br><br>Content-Type: application/xml<br><br>Content-Transfer-Encoding: 7bit<br><br>This is test2.<br><br>----abc-- |  Type: MIMEHierarchy<br>Type: Document<br>Value: This is test2 |

The business service needs the following properties on the child property set as shown in Table 15. These properties reflect the most accurate information on the data contained in the child property set.

Table 15. Properties for EAI MIME Doc Converter

| Property | Possible Values | Type | Description |
|---|---|---|---|
| ContentId | Any value | Document | No Default. The ContentId is the value used to identify the file attachment when the receiver parses the MIME message. When importing attachments, you should use a unique value for this property and not repeat it for the rest of the file attachments. This is required in the actual document. 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 15. 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)<br><br>octet-stream, pdf, postscript, x-tar, zip, x-javascript, msword, x-conference, x-gzip (used with ContentType of application)<br><br>aiff, basic, wav (used with ContentType of audio)<br><br>gif, jpeg, tiff, x-xbitmap (used with ContentType of image)<br><br>avi, mpeg (used with ContentType of video) | Document | Default is octet-stream. The ContentSubType value has to be specified if you want to set the content subtype of the document instead of using the extension to get a value from the MIME utility function. If the value is not provided the default value is used. |

**NOTE:** On the inbound direction, the business service is independent of the transport. It assumes that the input property set contains the MIME message and outputs a property set representation of the MIME message. A property set is used to represent each part of the MIME message. When decoding the MIME message, the business service automatically sets the properties based on the values in the MIME message.

# 8 Siebel Virtual Business Components

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

## About Virtual Business Components

A virtual business component (VBC) provides a way to access data that resides in an external data source using a Siebel business component. The VBC does not map to an underlying table in the Siebel Database. You create a new VBC in Siebel Tools and compile it into the siebel.srf file. The VBC calls a Siebel business service to provide a transport mechanism.

You can take two approaches to use virtual business components, as illustrated in Figure 34.

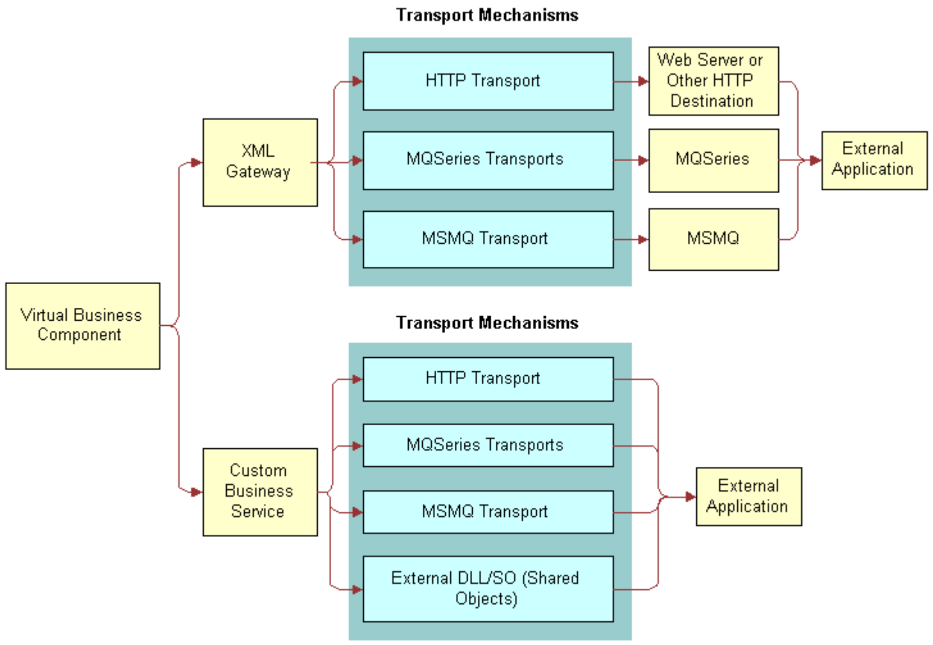


Figure 34. Two Approaches to Building Virtual Business Components

- Use the XML Gateway business service to pass data between the virtual business component and one of the Siebel transports, such as the EAI HTTP Transport, the EAI MQSeries AMI Transport, or the EAI MSMQ Transport.
- Write your own business service in Siebel eScript or in Siebel VB to implement the methods described in this chapter.

## About Using VBCs for Your Business Requirements

The following features enhance the functionality of VBCs to better assist you in meeting your business requirements:

- Virtual business components (VBCs) support drill down from a VBC. You can drill down to a VBC from a standard BC, another VBC, or the same VBC.
- A parent applet can be based on a VBC.
- You can define virtual business components that can participate as a parent in a business object. The VBC you define can be a parent to a standard BC or a VBC.
- You still can use an older version of XML format or property set by setting the VBC Compatibility Mode parameter to the appropriate version. For details, see Table 16 on page 136.
- You can pass search and sort specifications to the business service used by a VBC.

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

## Usage and Restrictions of VBCs

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

# Using Virtual Business Components

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

- ❑ Create a new Virtual Business Component.

  For details, see "Creating a New Virtual Business Component."

| | |
|---|---|
| ❑ | Set the User Properties on Virtual Business Components (VBCs).<br>For details, see "Setting User Properties for the Virtual Business Component" on page 136. |
| ❑ | Configure your VBC Business Service:<br>■ Configure your XML Gateway Service or write your own Business Service.<br>For details, see "XML Gateway Service" on page 138 and "Custom Business Service Methods" on page 150.<br>■ Configure your external application.<br>For details, see "External Application Setup" on page 150. |

## Creating a New Virtual Business Component

You create a new virtual business component in Siebel Tools.

***To create a new virtual business component***

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

## Setting User Properties for the Virtual Business Component

When defining the virtual business component, you must provide the user properties shown in Table 16.

Table 16. 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 16. Setting Virtual Business Component User Properties

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

***To define user properties***

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

**NOTE:** For list of different property sets and their format, see "Examples of Outgoing XML Format" on page 141 and "Examples of Incoming XML Format" on page 147.

## XML Gateway Service

The XML Gateway business service communicates between Siebel applications and external data sources using XML as the data format. For details on XML format, see "Examples of Outgoing XML Format" on page 141 and "Examples of Incoming XML Format" on page 147. The XML Gateway business service can be configured to use one of the following transports:

- EAI MQSeries AMI Server Transport
- EAI MQSeries Server Transport
- EAI HTTP Transport
- EAI MSMQ Transport

You can configure the XML Gateway by specifying the transport protocol and the transport parameters you use in the Service Parameters User Property of the virtual business component as shown in Table 17. When using the XML Gateway, you need to specify the following user properties for your virtual business component.

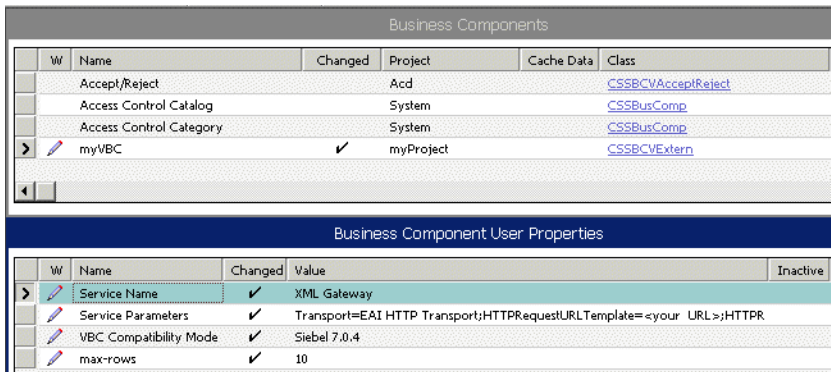
Table 17. User Properties

| Name | Value |
|---|---|
| Service Name | XML Gateway |
| Service Parameters | *variable1 name*=*variable1 value*;<br>*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 should not use any spaces between the name, the equal sign, the value, and the semicolon.

For example, if you want to specify the EAI HTTP Transport, you may use something like the following which is also illustrated in Figure 35:

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



Business Components

| W | Name | Changed | Project | Cache Data | Class |
|---|---|---|---|---|---|
| | Accept/Reject | | Acd | | CSSBCVAcceptReject |
| | Access Control Catalog | | System | | CSSBusComp |
| | Access Control Category | | System | | CSSBusComp |
| ✎ | myVBC | ✓ | myProject | | CSSBCVExtern |

Business Component User Properties

| W | Name | Changed | Value | Inactive |
|---|---|---|---|---|
| ✎ | Service Name | ✓ | XML Gateway | |
| ✎ | Service Parameters | ✓ | Transport=EAI HTTP Transport;HTTPRequestURLTemplate=<your URL>;HTTPR | |
| ✎ | VBC Compatibility Mode | ✓ | Siebel 7.0.4 | |
| ✎ | max-rows | ✓ | 10 | |

Figure 35. Setting Virtual Business Component User Properties

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

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

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

### *To implement VBC with MQSeries*

1. Call the EAI Business Integration Manager (Server Request) business service.
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 18.

Table 18. 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 19.

Table 19. XML Gateway Arguments

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

## Examples of Outgoing XML Format

Examples of the XML documents generated and sent by the XML Gateway to the external system are presented in Table 20. These examples are based on the example in "Custom Business Service Example" on page 165. See Appendix C, "DTDs for XML Gateway Business Service," for examples of the DTDs that correspond to each of these methods.

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

Table 20. Outgoing XML Tags and Descriptions

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

Table 20. Outgoing XML Tags and Descriptions

| Method | Format of the XML Stream | Description |
|---|---|---|
| Insert Request | `<siebel-xmlext-insert-req>`<br>`   <buscomp id="1">Contact</ buscomp>`<br>`   <remote-source>http://throth/ servlet/VBCContacts</remote- source>`<br>`   <row>`<br>`      <value field="AccountId">1- 6</value>`<br>`      <value field="Name">Max Adams</value>`<br>`      <value field="Phone">(398)765-1290</ value>`<br>`      <value field="Location">Troy</value>`<br>`      <value field="AccessId"></ value>`<br>`   </row>`<br>`</siebel-xmlext-insert-req>` | **siebel-xmlext-Insert-req.** This tag requests the commit of a new record in the remote system.<br>The insert-req XML stream contains values for fields entered through the business component. |
| PreInsert Request | `<siebel-xmlext-preinsert-req>`<br>`   <buscomp id="1">Contact</ buscomp>`<br>`   <remote-source>http://throth/ servlet/VBCContacts</remote- source>`<br>`</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 20. Outgoing XML Tags and Descriptions

| Method | Format of the XML Stream | Description |
|---|---|---|
| Query Request | `<siebel-xmlext-query-req>`<br>`<buscomp id="1">Contact</buscomp>`<br>`<remote-source>http://throth/servlet/VBCContacts</remote-source>`<br>`<max-rows>6</max-rows>`<br>`<search-string>=([Phone] IS NOT NULL) AND ([AccountId] = "1-6")</search-string>`<br>`<search-spec>`<br>`<node node-type="Binary Operator">AND`<br>`<node node-type="Unary Operator">IS NOT NULL`<br>`<node node-type="Identifier">Phone</node>`<br>`</node>`<br>`<node node-type="Binary Operator">=`<br>`<node node-type="Identifier">AccountId</node>`<br>`<node value-type="TEXT" node-type="Constant">1-6</node>`<br>`</node>`<br>`</node>`<br>`</search-spec>`<br>`<sort-spec>`<br>`<sort field="Location">ASCENDING</sort>`<br>`<sort field="Name">DESCENDING</sort>`<br>`</sort-spec>`<br>`</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.<br><br>**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.<br><br>**search-string.** The search specification used to query and filter the information.<br><br>**search-spec.** Hierarchical representation of the search-string. For details, see "Search-Spec Node-Type Types" on page 146.<br><br>**sort-spec.** List of sort fields and sort order. |

Table 20. Outgoing XML Tags and Descriptions

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

## Search-Spec Node-Type Types

The search-string is in the Siebel query language format. The search-string is parsed by the Siebel query object and then turned into the hierarchical search-spec. Table 21 shows the different search-spec node-types and their values.

Table 21. Search-Spec Node-Types

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

## Examples of Incoming XML Format

Table 22 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 165. See Appendix C, "DTDs for XML Gateway Business Service," for examples of the DTDs that correspond to each of these methods.

Table 22. 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>`<br><br>`<status-code>4</code>`<br><br>`<error-field>Name</error-field>`<br><br>`<error-text>Name must not be empty</error-text>`<br><br>`</siebel-xmlext-status>` | Format of the XML stream expected by the Siebel application in case of an error in the external application. The tags for this XML stream, including the entire XML stream, are optional. If the error is specific to a field, the field name should be specified.<br><br>**siebel-xmlext-status.** This tag is used to check the status returned by the external system.<br><br>**status-code.** This tag overrides the return value.<br><br>**error-text.** This tag specifies textual representation of the error, if it is available. This tag appears in addition to the standard error message. For example, if Siebel application attempts to update a record in the external system with a NULL Name, and this is not allowed in the external system, then the error text is set to "Name must not be empty." |

Table 22. Incoming XML Tags and Descriptions

| Method | Format of the XML Stream | Description |
|---|---|---|
| Init Return | `<siebel-xmlext-fields-ret>`<br>`   <support field="AccountId"/>`<br>`   <support field="Name"/>`<br>`   <support field="Phone"/>`<br>`   <support field="Location"/>`<br>`   <support field="AccessId"/>`<br>`</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.<br>The following field names are reserved by the Siebel application and should not appear in this list:<br>Id, Created, Created By, Updated, Updated By. |
| Insert Return | `<siebel-xmlext-insert-ret>`<br>`   <row>`<br>`     <value field="AccountId">1-6</value>`<br>`      <value field="Name">Max Adams</value>`<br>`     <value field="Phone">(398)765-1290</value>`<br>`     <value field="Location">Troy</value>`<br>`      <value field="AccessId">146</value>`<br>`   </row>`<br>`</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>`<br>`   <row>`<br>`      <value field="Location">San Jose</value>`<br>`   </row>`<br>`</siebel-xmlext-preinsert-ret>` | **siebel-xmlext-preinsert-ret.** Returns default values for each field, if there is any default value. |

Table 22. Incoming XML Tags and Descriptions

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

Table 22. Incoming XML Tags and Descriptions

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

## External Application Setup

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

## Custom Business Service Methods

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

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

These methods pass property sets between the virtual business component and the business service. Virtual business component methods take property sets as arguments. Each method takes two property sets: an Inputs property set and an Outputs property set. The methods are called by the *CSSBCVExtern* class in response to requests from other objects that refer to or are based on the virtual business component.

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

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

You may also choose to use the XML Gateway service to allow the connection for your VBC. For details, see "XML Gateway Service" on page 138.

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

## Common Method Parameters

Table 23 shows the input parameters common to every method. Please note that all these parameters are at the root property set.

Table 23. Common Input Parameters

| Parameter | Description |
|---|---|
| Remote Source | Optional. Specifies the name of an external data source. This is the VBC's Remote Source user property, if defined. For details, see Table 16 on page 136. |
| Business Component Name | Name of the active virtual business component. |
| Business Component Id | Internally generated unique value that represents the virtual business component. |
| Parameters | Optional. The VBC's Service Parameters user property, if defined. For details, see Table 16 on page 136. A set of parameters required by the business service. |
| VBC Compatibility Mode | Optional. This is the VBC's Compatibility Mode user property, if defined. For details, see Table 16 on page 136. |

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

## Business Services Methods and Their Property Sets

The following examples display each method's input and output property sets for a virtual business component Contact that displays simple contact information for a given account. These examples are based on the example in the "Custom Business Service Example" on page 165.

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

### Delete

The Delete method is called when a record is deleted. Figure 36 illustrates the property set for the Delete input and is followed by its XML representation.

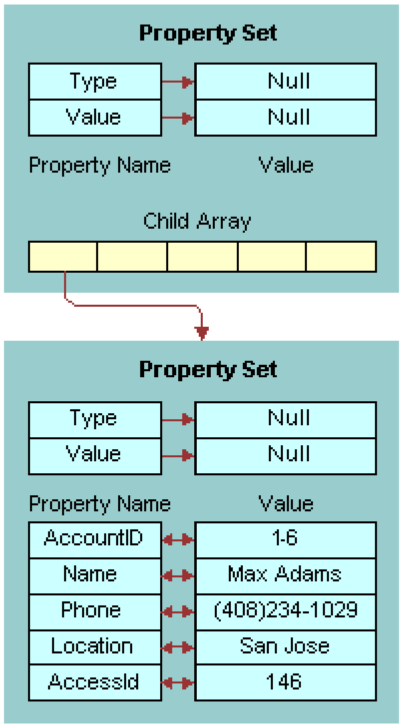


Figure 36. Delete Input Property Set

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

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

<PropertySet

   Business_spcComponent_spcId="1"

   Business_spcComponent_spcName="Contact">

   <PropertySet

      AccountId="1-6"

      Name="Max Adams"

      Phone="(408)234-1029"

      Location="San Jose"

      AccessId="146" />
```

```
</PropertySet>
```

Figure 37 illustrates the property set for Delete output and is followed by its XML representation.

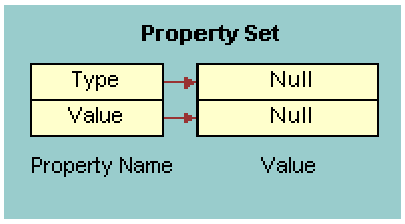


Figure 37.  Delete Output Property Set

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

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

<PropertySet />
```

### Error Return

Figure 36 illustrates the property set for the Error Return, when an error is detected. The illustration is followed by its XML representation.

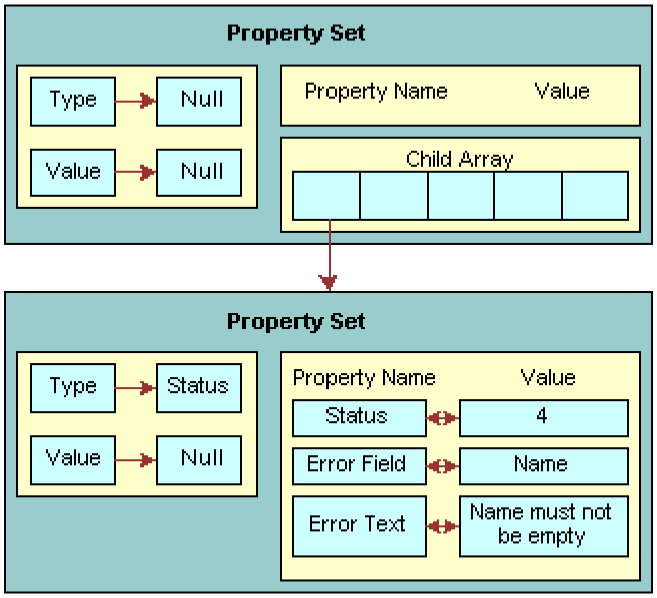


Figure 38.  Error Return Property Set

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

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

```
<PropertySet>

   <Status Status="4"

      Error_spcField="Name"

      Error_spcText="Name must not be empty"/>

</PropertySet>
```

### Init

The Init method is called when the virtual business component is first instantiated. It initializes the virtual business component. It expects to receive the list of fields supported by the external system. Figure 39 illustrates the property set for Init input and is followed by its XML representation.

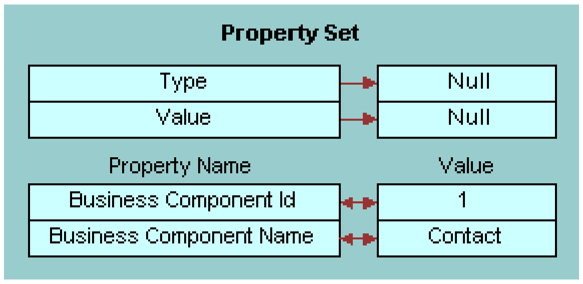


Figure 39. Init Input Property Set

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

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

<PropertySet

   Business_spcComponent_spcId="1"

   Business_spcComponent_spcName="Contact"/>
```

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

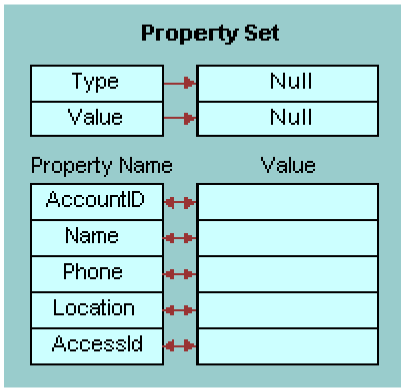


Figure 40.  Init Output Property Set

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

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

<PropertySet

   AccountId=""

   Name=""

   Phone=""

   Location=""

   AccessId="" />
```

## Insert

The Insert method is called when a New Record is committed. Figure 41 illustrates the property set for Insert input and is followed by its XML representation.

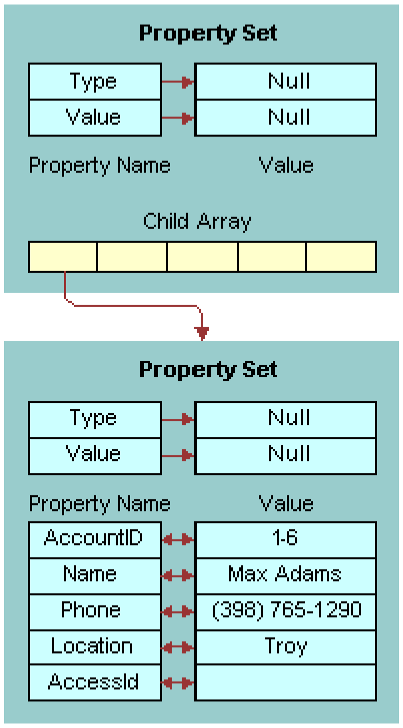


Figure 41. Insert Input Property Set

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

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

<PropertySet

   Business_spcComponent_spcId="1"

   Business_spcComponent_spcName="Contact">

   <PropertySet

      AccountId="1-6"

      Name="Max Adams"

      Phone="(398)765-1290"

      Location="Troy"

      AccessId="" />
```

```
</PropertySet>
```

Figure 42 illustrates the property set for Insert output and is followed by its XML representation.

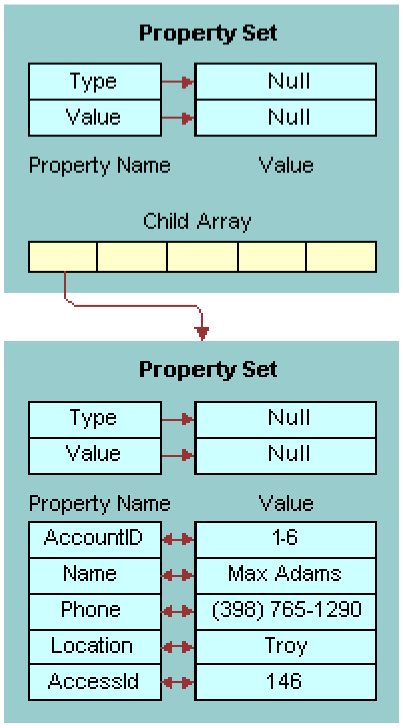


Figure 42. Insert Output Property Set

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

### PreInsert

The PreInsert method is called when a New Record operation is performed. It supplies default values. Figure 43 illustrates the property set for PreInsert input and is followed by its XML representation.

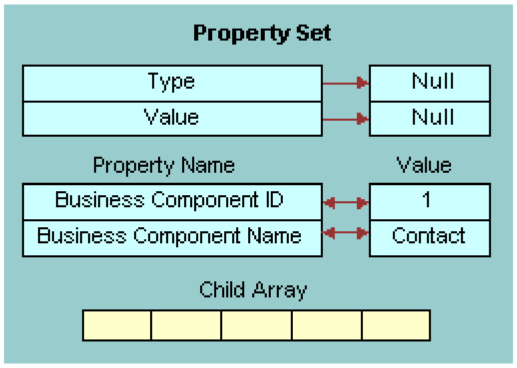


Figure 43. PreInsert Input Property Set

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

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

<PropertySet

   Business_spcComponent_spcId="1"

   Business_spcComponent_spcName="Contact"/>
```

Figure 44 illustrates the property set for PreInsert output and is followed by its XML representation.

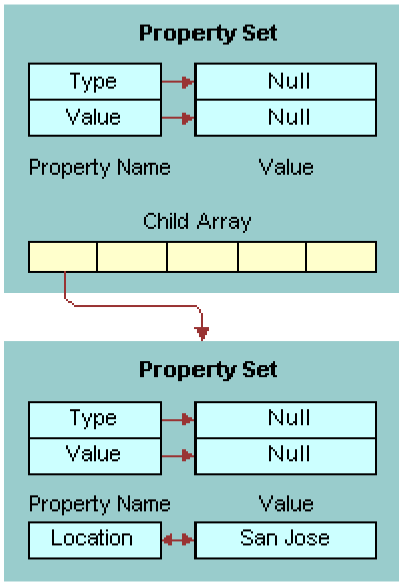


Figure 44. PreInsert Output Property Set

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

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

<PropertySet>

   <PropertySet Location="San Jose" />

</PropertySet>
```

## Query

The Query method is called when a search is performed. The Query method must be supported by every virtual business component. Each record that matches the query is represented as a property set. For example, if 5 records match the query, there will be 5 child property sets. Each property set will contain a list of field names—field value pairs representing the values of each field for that particular record. Figure 46 illustrates the property set for Query input and is followed by its XML representation.

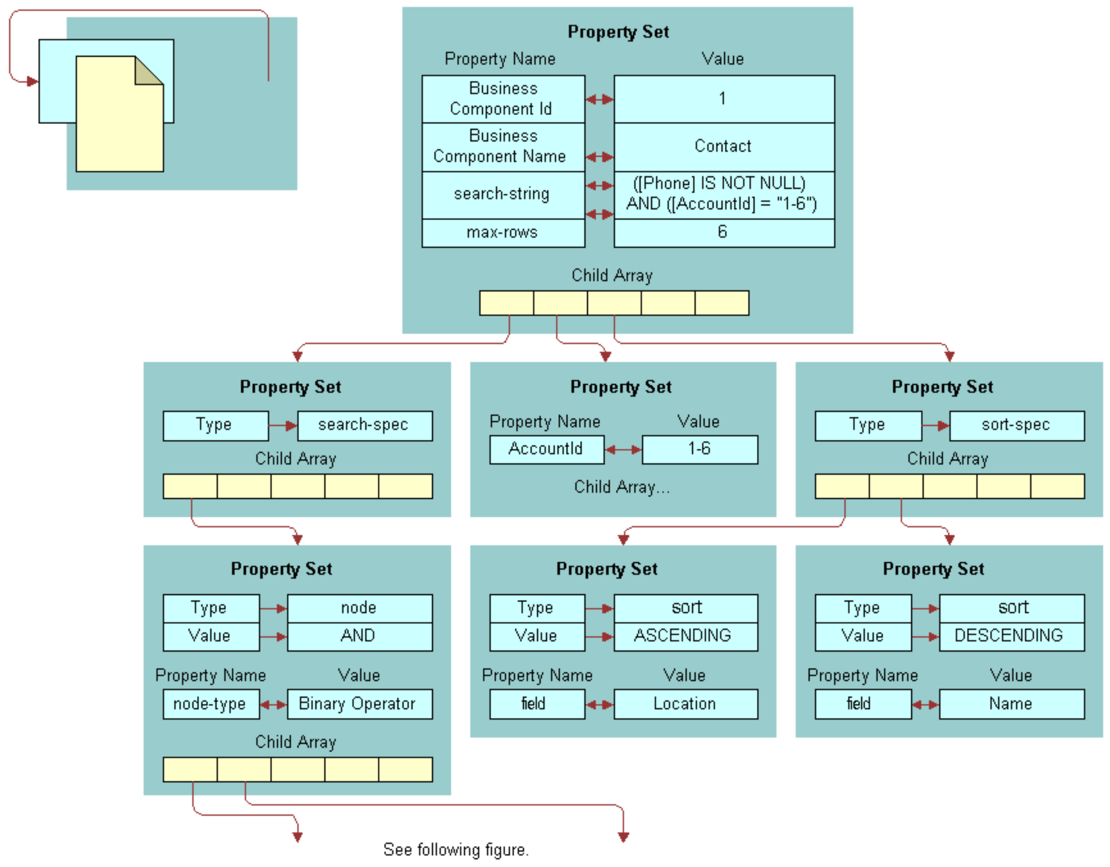


Figure 45. Query Input Property Set (Part 1)

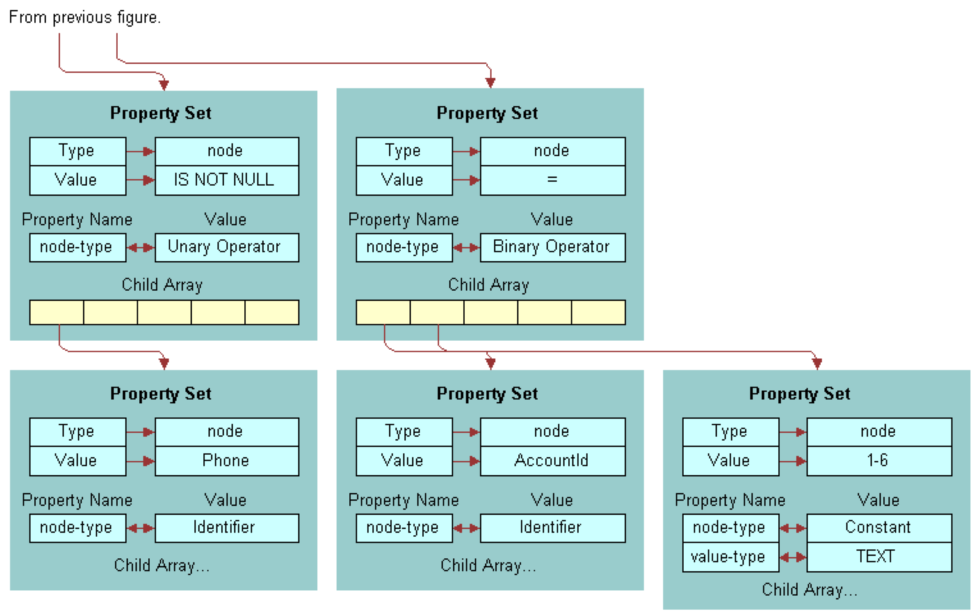


Figure 46. Query Input Property Set (Part 2)

```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet
   max-rows="6"
   search-string="([Phone] IS NOT NULL) AND ([AccountId] = "1-6")"
   Business_spcComponent_spcId="1"
   Business_spcComponent_spcName="Contact">
   <PropertySet AccountId="1-6" />
      <search-spec>
         <node node-type="Binary Operator">AND
         <node node-type="Unary Operator">IS NOT NULL
         <node node-type="Identifier">Phone</node>
            </node>
            <node node-type="Binary Operator">=
            <node node-ype="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 47 illustrates the property set for Query output and is followed by its XML representation.

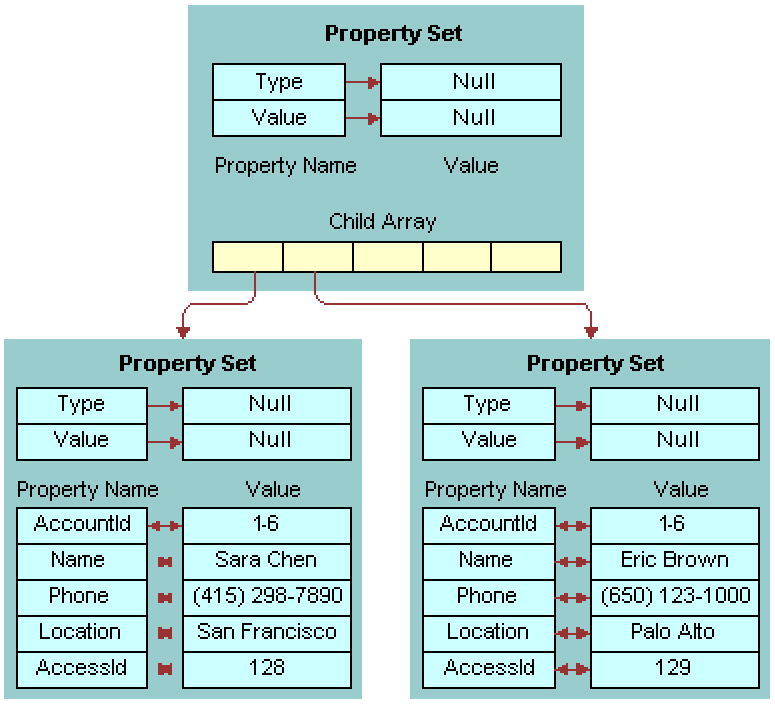


Figure 47. Query Output Property Set

```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet>
   <PropertySet
      AccountId="1-6"
      Name="Sara Chen"
      Phone="(415)298-7890"
      Location="San Francisco"
      AccessId="128" />
   <PropertySet
      AccountId="1-6"
      Name="Eric Brown"
      Phone="(650)123-1000"
      Location="Palo Alto"
      AccessId="129" />
</PropertySet>
```

### Update

The Update method is called when a record is modified. Figure 48 illustrates the property set for Update input and is followed by its XML representation.

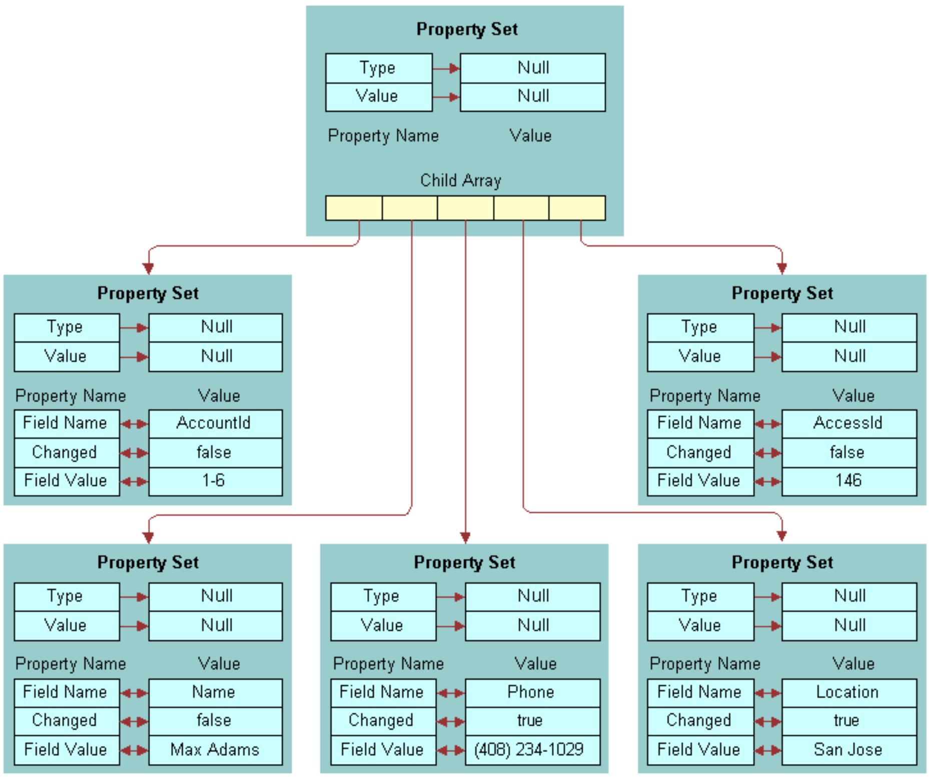


Figure 48. Update Input Property Set

```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet
   Business_spcComponent_spcId="1"
   Business_spcComponent_spcName="Contact">
   <PropertySet
      Field_spcName="AccountId"
      Changed="false"
      Field_spcValue="1-6" />
   <PropertySet
      Field_spcName="Name"
      Changed="false"
      Field_spcValue="Max Adams" />
   <PropertySet
      Field_spcName="Phone"
      Changed="true"
      Field_spcValue="(408)234-1029" />
   <PropertySet
      Field_spcName="Location"
      Changed="true"
      Field_spcValue="San Jose" />
   <PropertySet
      Field_spcName="AccessId"
      Changed="false"
      Field_spcValue="146" />
</PropertySet>
```

Figure 49 illustrates the property set for the Update output and is followed by its XML representation.

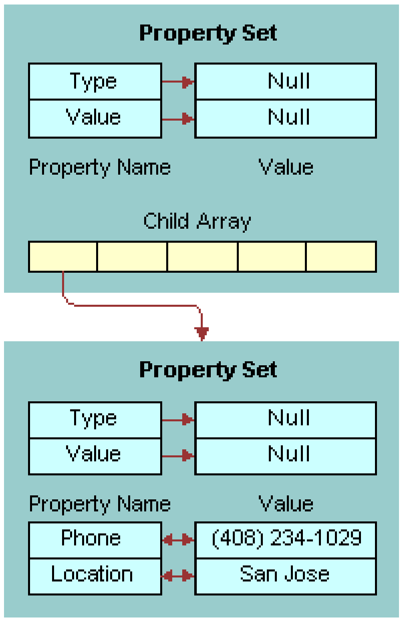


Figure 49. Update Output Property Set

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

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

<PropertySet

   <PropertySet

      Phone=="(408)234-1029"

      Location="San Jose" />

</PropertySet>
```

## Custom Business Service Example

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

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

```
    }
    else if (MethodName == "Query") {
        return(Query(Inputs, Outputs));
    }
    else if (MethodName == "PreInsert") {
        return(PreInsert(Inputs, Outputs));
    }
    else if (MethodName == "Insert") {
        return(Insert(Inputs, Outputs));
    }
    else if (MethodName == "Update") {
        return(Update(Inputs, Outputs));
    }
    else if (MethodName == "Delete") {
        return(Delete(Inputs, Outputs));
    }
    else {
        return (ContinueOperation);
    }
}
function Init (Inputs, Outputs)
{
    // For debugging purpose...
    logPropSet(Inputs, "InitInputs.xml");
    Outputs.SetProperty("AccountId", "");
    Outputs.SetProperty("Name", "");
    Outputs.SetProperty("Phone", "");
    Outputs.SetProperty("AccessId", "");
    Outputs.SetProperty("Location", "");
    // For debugging purpose...
    logPropSet(Outputs, "InitOutputs.xml");
    return (CancelOperation);
}
function Query(Inputs, Outputs)
{
    // For debugging purpose...
    logPropSet(Inputs, "QueryInputs.xml");
    var selectStmt = "select * from Contacts ";
    var whereClause = " where ";
    var orderbyClause = " order by ";
    // You have the following properties if you want to use them
    // Inputs.GetProperty("Business Component Name")
    // Inputs.GetProperty("Business Component Id")
    // Inputs.GetProperty("Remote Source")
    // If you configured Maximum Cursor Size at the buscomp,
    // get max-rows property
    var maxRows = Inputs.GetProperty("max-rows");
    // get search-string
    var searchString = Inputs.GetProperty("search-string");
    // convert the search-string into a where clause
    searchString = stringReplace(searchString, '*', '%');
    searchString = stringReplace(searchString, '[', ' ');
    searchString = stringReplace(searchString, ']', ' ');
    searchString = stringReplace(searchString, '~', ' ');
    searchString = stringReplace(searchString, '"', "'");
```

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

```
        {
        var fieldValue = fields.Item(j).Value();
        if (fieldValue == null)
            row.SetProperty(fields.Item(j).Name(), "");
        else
            row.SetProperty(fields.Item(j).Name(), fieldValue);
        }
        Outputs.AddChild(row);
        rs.MoveNext();
    }
    // For debugging purpose...
    logPropSet(Outputs, "QueryOutputs.xml" );
    // clean up
    child = null;
    sortProp = null;
    row = null;
    rs.Close();
    rs = null;
    conn.Close();
    conn = null;
    return (CancelOperation);
}
function PreInsert (Inputs, Outputs)
{
    // For debugging purpose...
    logPropSet(Inputs, "PreInsertInputs.xml");
    var defaults = TheApplication().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;
}
```

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

# 9 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:

## Configuring External Business Components

To configure external business components, you perform the high-level tasks described below. Before proceeding, review "Configuring the Business Component" on page 180.

- Create a table definition.

  Import the external table definition into Siebel Tools using the External Table Schema Import Wizard.

  This wizard creates a new Table object definition in the Siebel Repository, based upon the contents of a DDL (data definition language) file you provide.

  As may be appropriate, you can import an external view definition rather than a table definition. References to an external table also implies an external view, according to your implementation.

- Map columns in the external table to Siebel system fields.
- Configure the external business component and specify the data source object.
- Create or update a business component to utilize the new Table object definition.
- Specify run-time parameters.
  - Configure the new data source in the Server Administration view.
  - Add the data source to the *OM - Named Data Source name* component parameter.

## Creating the External Table Definition

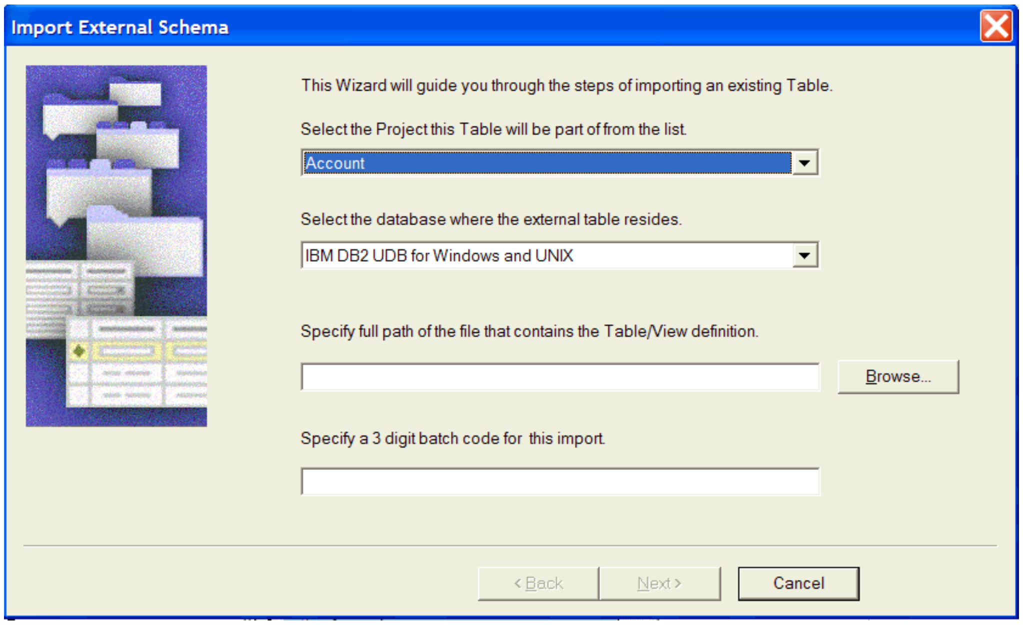
You use Siebel Tools and the External Table Schema Import Wizard to import your external table definition into the Siebel Repository.

For more information about using Siebel Tools, see *Using Siebel Tools* on the *Siebel Bookshelf*.

***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, double-click External Table Schema Import.

   The External Table Schema Import Wizard appears, as shown in the following figure.



Import External Schema

This Wizard will guide you through the steps of importing an existing Table.

Select the Project this Table will be part of from the list.

Account

Select the database where the external table resides.

IBM DB2 UDB for Windows and UNIX

Specify full path of the file that contains the Table/View definition.

Specify a 3 digit batch code for this import.

5. In the External Table Schema Import Wizard, specify values as described below:
   - The project the new Table object definition will be associated with.
   - The database where the external table resides. The value specified should 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 will allow 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 need 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 24 contains the data type mappings you can use when importing table definitions.

Table 24. 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 |
| FLOAT | Numeric |
| REAL | Numeric |

Table 24. Supported Data Type Mappings by Product

| Supported Data Types | Siebel Data Type |
|---|---|
| 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 |
| **Siebel Analytics Data Types** | |
| Integer | Numeric with scale of 0 |
| Smallint | Numeric with scale of 0 |
| Tinyint | Numeric with scale of 0 |
| Float | Numeric |
| Double | Numeric |
| Bit | Character (1) |
| Boolean | Character (1) |
| Char | Character |
| Varchar | Varchar |
| Longvarchar | Long |

Table 24. Supported Data Type Mappings by Product

| Supported Data Types | Siebel Data Type |
|---|---|
| Datetime | Datetime |
| Date | Datetime |
| Time | Datetime |

Table 25 contains the data types that are not supported for importing table definitions.

Table 25. Unsupported Data Type Mappings by Product

| Unsupported Data Types |
|---|
| **MS SQL Server Data Types** |
| timestamp |
| varbinary |
| binary |
| image |
| cursor |
| uniqueidentifier |
| **DB2 Universal Database Data Types** |
| CLOB |
| DBCLOB |
| BLOB |
| **Oracle Data Types** |
| TIMESTAMP |
| CLOB |
| NCLOB |
| BLOB |
| BFILE |
| ROWID |
| UROWID |
| RAW |
| LONG RAW |
| INTERVAL YEAR TO MONTH |
| INTERVAL DAY TO SECOND |

Table 25. Unsupported Data Type Mappings by Product

| Unsupported Data Types |
|---|
| **Siebel Analytics Data Types** |
| 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.

## Specify Additional Table Properties

Once 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*.

## Mapping External Columns to Siebel System Fields

Once the external business component has been defined, you must map the Siebel system fields to the corresponding external table column. Starting with Siebel 7.7, System Field mapping is accomplished at the column definition rather than using business component user properties. You need to specify the System Field Mapping column attribute if you wish 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, Siebel system fields will be not be included in the generated SQL for external tables.

System Field Mapping is used to specify the mapping between table columns and Siebel internal fields.

Following is a list of the Siebel 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

Once the external table has been defined, you will need to specify the data source for the corresponding external table. A new child object, the Data Source, is added to the Table object in Siebel 7.7.

- The data source object corresponds to a data source defined in the application configuration file (.cfg) or in the Application - Server Configuration > Profile Configuration view.
- The data source object instructs the Application Object Manager to use the data source for a specific table. If a Data Source is not specified, the default data source for the application will be used.

**NOTE:** The Data Source object should be specified for external tables only.

For more information about the data source child object, see *Siebel Tools Online Help*.

## Configuring the Business Component

Once a Table object definition corresponding to the external table exists in the repository, you can configure a business component to utilize the new Table object definition.

In general, configuring an external business component is similar to configuring a standard business component with the following exceptions:

- The Data Source business component property should be 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 should be 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 must reside in the same database instance as the child table.
- It is recommended that all external business components 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 eBusiness Applications.

## Specifying Run-Time Parameters

After the data source definition is named in Siebel Tools, you will specify the run-time parameters by creating a Data Source definition in the Administration - Server Configuration screen and updating the Server Component definition.

***To create the Data Source definition in the Administration - Server Configuration screen***

1. Navigate to the Administration - Server Configuration screen and select the Enterprises amazon link.
2. From the Enterprise Server, view select the Profile Configuration tab.
3. Copy an existing record that has a Subsystem Type of InfraDatasources.
4. Change the Profile and Alias properties to the Data Source name configured in Siebel Tools.
5. Update the profile parameters to correspond to the external RDBMS:

   DSConnectString = the data source connect string.

   DSSSQLStyle = database SQL type.

   DSTableOwner = data source table owner.

   DSUsername = default user name used for connections.

   DSPassword = default password used for connections.

   **NOTE:** The DSUsername and the DSPassword parameters are optional. If specified, this will override the default user name and password.

## 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 26 contains the supported connector DLL names and the corresponding SQL styles.

Table 26. Supported Connector DLL Names and SQL Styles

| DLL Names | SQL Styles |
|---|---|
| sscdo90.dll | Oracle |
| sscdo90.dll | OracleCBO |
| sscdm80.dll | MSSqlServer |
| sscddcli.dll | DB2 |
| sscdw8.dll | Watcom (an SQL Anywhere database) |
| sscdsacon.dll | Siebel Analytics Server |

***To update the Server Component to use the new data source***

1. Navigate to the Administration - Server Configuration screen and select Enterprises from the link bar.
2. From the Enterprise Servers view, select the Component Definitions tab.
3. In the Component Definitions list applet, select your Application Object Manager Component. For example, select the Call Center Object Manager (ENU).

4. 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. You will need to reselect your application component after selecting the Start Reconfiguring menu item.

5. 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 create the Data Source definition in the Administration - Server Configuration screen" section.

   The format of the OM - Named Data Source name parameter is a comma-delimited list of data source aliases. It is recommended that the default values not be modified and that you append their new data sources to the preexisting list.

6. 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 will be 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

Following are 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 will return 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, that should 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 write. This is typically used by EBCs 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 into 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 external business components:

- Creating and populating the external table is the responsibility of the customer. Consult your database administrator for the appropriate method to use.
- External business components cannot be docked, so they do not apply to mobile users on the Siebel Mobile Web Client. Siebel Remote is not supported.
- External business components support many-to-many relationships with the limitation that for such relationships the intersection table needs to be from the same data source as the child business component.
- External business components cannot be loaded using Enterprise Integration Manager.
- Siebel Server components other than an Object Manager (for example Assignment Manager, Workflow Policies, Incentive Compensation, and so on) cannot utilize external business components.
- The Id field must be mapped to an underlying column in the external table in order to support insert, update, delete, and query operations.

- For external business components that contain multi-value groups, if a primary join is enabled, then both the parent and the child business components need to 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 external business components.
- An external join alias must have the same name as the name used for the external table.

**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 will appear 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 DB Multiplex - Max Number of Shared DB Connections (DSMaxSharedDbConns), the DB Multiplex - Min Number of Shared DB Connections (DSMinSharedDbConns), and the 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 external business component are read-only.

The limitations for joining business components to tables in an external data source follow:

- 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 will decide 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 motivated by the desire to:

1. Retrieve the first few rows quickly.
2. Ship the least amount of data between the Siebel and external data sources.
3. Eliminate a sort step.

Step 2 and Step 3 may develop into 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 will raise an error if more than 1000 rows are retrieved. The query specification should be refined 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 very selective. If there is a search specification on Field A, the table that field A is based on should be strongly considered as 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 should be strongly considered as 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 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 the external business components).

  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 would 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 default 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 distributed join.
- Inner join is not supported for distributed join.
- Reverse navigation (for example, call to go to the last record) is not supported when 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 a Siebel Analytics Presentation Folder for Use as an External Table

An external business component 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.

External business components, in conjunction with the Analytics database connector, allow the ability to construct business components that derive their data from Siebel Analytics. External business components support Siebel Analytics as a source for having Siebel Tools import the structure of a Siebel Analytics Presentation Folder by reading sources such as an XML file.

**NOTE:** Analytics integration is read-only, and any business components that utilize Analytics as a data source must be configured to support read-only access.

The following instructions generate the Siebel Analytics Presentation folder as an XML file from Siebel Analytics.

### *To load a Siebel Analytics presentation folder for use as an external table*

1. Start the Siebel Analytics Administration Tool.
2. Select File > New and create a new repository file.
3. Select File > Import from Repository.
   a. Choose the appropriate repository and click Next.
   b. When requested, type the log in and password.
   c. Select the object Catalog from the drop-down list.
   d. Choose the catalog you wish to import into Siebel Tools and click Add With Children.
   e. Click Next.
   f. Click Finish.
4. Select Tools > Utilities, choose Repository Documentation and click Execute….
5. In the Save as Type field, select XML as the file extension.
6. Give a new name to the file and click Save.

Follow these instructions to avoid the Repository Documentation wizard exporting the full repository definition into an XML file, and not only the selected object.

If you import an XML file that contains several presentation folders, Siebel Tools creates one external table per presentation folder.

# A Predefined EAI Business Services

Siebel eBusiness Applications provide a number of business services. These services do not require any modification, but they do require that you choose and configure them to suit your requirements.

**NOTE:** For general information on using business services, refer to Chapter 4, "Business Services."

Table 27 presents the predefined Siebel eAI business services.

Table 27. Predefined EAI Business Services

| Business Service | Class | Description |
|---|---|---|
| EAI XSD Wizard | | Used to create integration objects based on XSD files. |
| EAI XML XSD Generator | | Used to generate an XSD file from an integration object. |
| EAI Transaction Service | CSSBeginEndTransactionService | EAI Transaction service for working with Siebel transactions such as begin, end, or find out whether in transaction. |
| EAI MSMQ Transport | CSSMsmqTransService | EAI MSMQ Transport. |
| EAI MQSeries Server Transport | CSSMqSrvTransService | EAI MQSeries Server Transport. |
| EAI MQSeries AMI Transport | CSSMqAmiTransService | EAI MQSeries AMI Transport. For details, see *Transports and Interfaces: Siebel eBusiness Application Integration Volume III*. |
| EAI HTTP Transport | CSSHTTPTransService | EAI HTTP Outbound Transport. For details, see *Transports and Interfaces: Siebel eBusiness Application Integration Volume III*. |
| EAI Siebel Adapter | CSSEAISiebelAdapterService | EAI Siebel Adapter. For details, see Chapter 6, "EAI Siebel Adapter." |
| EAI Query Spec Service | CSSEAIQuerySpecService | Used internally by EAI Siebel Adapter to convert SearchSpec method argument as string to an Integration Object Instance that EAI Siebel Adapter can use as a Query By Example object. |

Table 27. Predefined EAI Business Services

| Business Service | Class | Description |
|---|---|---|
| EAI Import Export | CSSEAIImportExportService | EAI Import Export Service (import and export integration object from or to XML). |
| EAI BTS COM Transport | CSSEAIBtsComService | EAI Siebel to BTS COM Transport. |
| EAI DLL Transport | CSSDllTransService | EAI DLL Transport. For details, see *Transports and Interfaces: Siebel eBusiness Application Integration Volume III*. |
| EAI Data Mapping Engine | CSSDataTransformationEngine | EAI Data Transformation Engine. For details, see *Business Processes and Rules: Siebel eBusiness Application Integration Volume IV*. |
| No Envelope | CSSEAINullEnvelopeService | EAI Null Envelope Service. For details, see *XML Reference: Siebel eBusiness Application Integration Volume V*. |
| Siebel Message Envelope | CSSEAISMEnvelopeService | EAI Siebel Message Envelope Service. For details, see *XML Reference: Siebel eBusiness Application Integration Volume V*. |
| EAI Dispatch Service | CSSEAIDispatchService | Dispatch Service. For details, see *Business Processes and Rules: Siebel eBusiness Application Integration Volume IV*. |
| EAI Integration Object to XML Hierarchy Converter | CSSEAIIntObjHierCnvService | EAI Integration Object Hierarchy (also known as SiebelMessage) to XML hierarchy converter service. For details, see *XML Reference: Siebel eBusiness Application Integration Volume V*. |
| EAI MIME Hierarchy Converter | CSSEAIMimePropSetService | EAI MIME Hierarchy Conversion Service. For details, see Chapter 7, "Siebel eAI and File Attachments." |
| EAI MIME Doc Converter | CSSEAIMimeService | MIME Document Conversion Service. For details, see Chapter 7, "Siebel eAI and File Attachments." |
| EAI XML Converter | CSSEAIXMLCnvService | Converts between XML and EAI Messages. For details, see *XML Reference: Siebel eBusiness Application Integration Volume V*. |

Table 27. Predefined EAI Business Services

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

# 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*.

## Property Sets and Integration Objects

Many eAI business services operate on integration object instances. Since business services take property sets as inputs and outputs, it is necessary to represent integration objects as property sets. The mapping of integration objects, components, and fields to property sets is known as the Integration Object Hierarchy.

Using this representation, you can pass a set of integration object instances of a specified type to an eAI business service. You pass the integration object instances as a child property set of the business service method arguments. This property set always has a type of SiebelMessage. You can pass the SiebelMessage property set from one business service to another in a workflow without knowing the internal representation of the integration objects.

## Property Set Node Types

When passing integration object instances as the input or output of a business service, you can use property sets to represent different node types, as presented in Table 28.

Table 28. Property Set Node Types

| Name | Parent | Value of Type Attribute | Properties | Description |
|---|---|---|---|---|
| Service Method Arguments | N/A | Ignored | The properties of this property set contain any service specific parameters, such as *PrimaryRowId* for EAI Siebel Adapter. | This is the top-level property set of a business service's input or output. The properties of this property set contain any service-specific parameters (for example, *PrimaryRowId* for EAI Siebel Adapter). |
| SiebelMessage | Service Method Arguments | SiebelMessage | The properties of this property set contain header attributes associated with the integration object, for example, *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 28. 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 of a Sample Account

This example shows an Account integration object in which the object has two component types: Account and Business Address (which is a child of Account). The hierarchy of component types from a Siebel Tools perspective, looks like that shown in Figure 50.

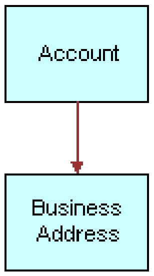


Figure 50. Sample Account Integration Object

Figure 51 on page 197 shows an example instance of this object type, using the Integration Object Hierarchy representation. There are two Sample Account instances. The first object instance has an Account component and two Business Address child components. The second object instance has only an Account component with no child components.

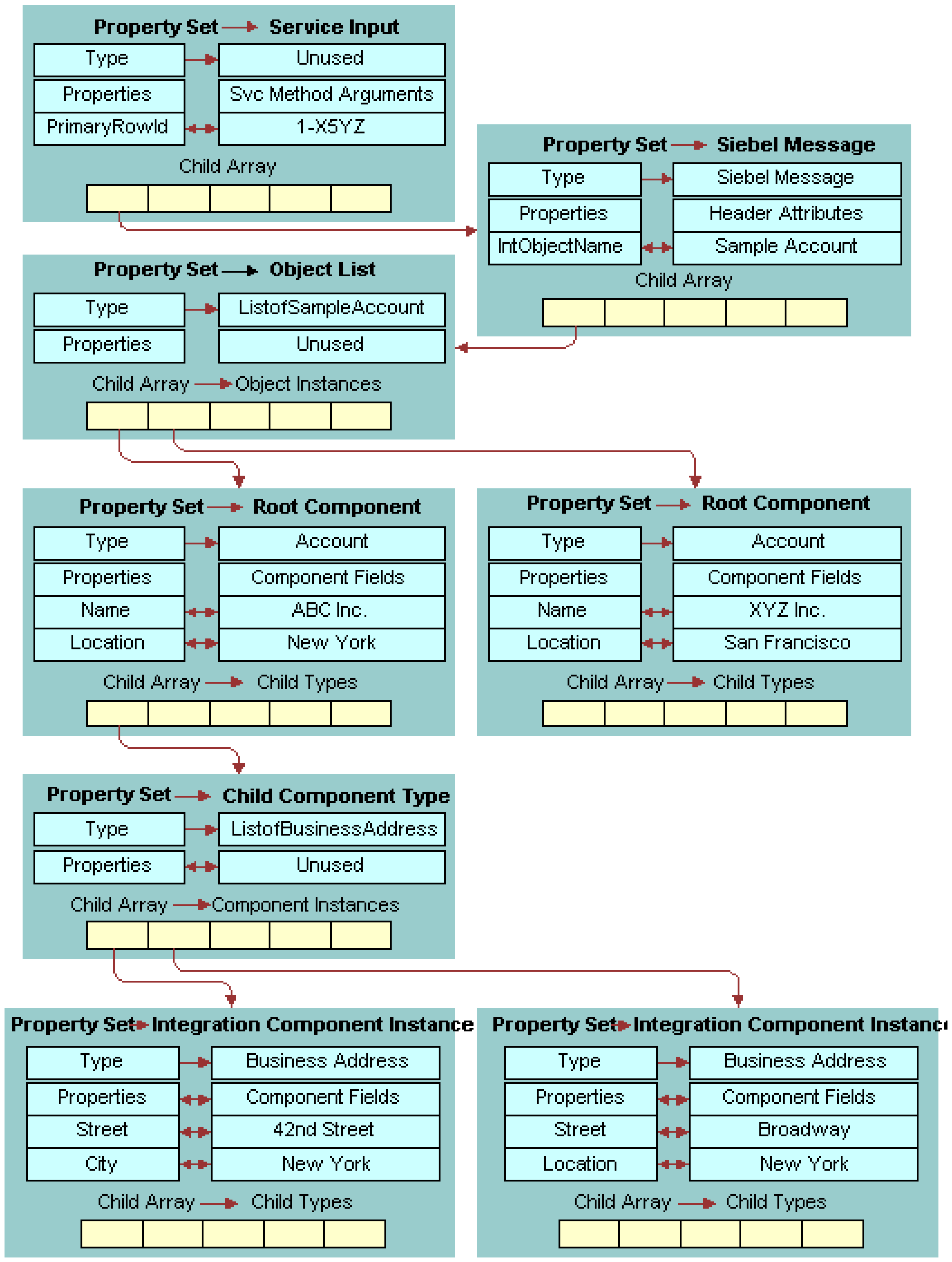

Property Set → Service Input
Type → Unused
Properties: Svc Method Arguments
PrimaryRowId ↔ 1-X5YZ
Child Array
Property Set → Siebel Message
Type → Siebel Message
Properties: Header Attributes
IntObjectName ↔ Sample Account
Child Array
Property Set → Object List
Type → ListofSampleAccount
Properties: Unused
Child Array → Object Instances
Property Set → Root Component
Type → Account
Properties: Component Fields
Name ↔ ABC Inc.
Location ↔ New York
Child Array → Child Types
Property Set → Root Component
Type → Account
Properties: Component Fields
Name ↔ XYZ Inc.
Location ↔ San Francisco
Child Array → Child Types
Property Set → Child Component Type
Type → ListofBusinessAddress
Properties ↔ Unused
Child Array → Component Instances
Property Set → Integration Component Instance
Type → Business Address
Properties ↔ Component Fields
Street ↔ 42nd Street
City ↔ New York
Child Array → Child Types
Property Set → Integration Component Instance
Type → Business Address
Properties ↔ Component Fields
Street ↔ Broadway
Location ↔ New York
Child Array → Child Types

Figure 51. 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.

## Outbound DTDs

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

### Delete

```
<!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

```
<!ELEMENT siebel-xmlext-fields-req (buscomp, remote-source?)>

<!ELEMENT buscomp (#PCDATA)>

<!ATTLIST buscomp id NMTOKEN #REQUIRED >

<!ELEMENT remote-source (#PCDATA)*>
```

### Insert

```
<!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

```
<!ELEMENT siebel-xmlext-preinsert-req (buscomp, remote-source?)>

<!ELEMENT buscomp (#PCDATA)>

<!ATTLIST buscomp id NMTOKEN #REQUIRED >

<!ELEMENT remote-source (#PCDATA)*>
```

## Query

```
<!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

```
<!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

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

### Delete Response

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

### Init Response

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

<!ELEMENT support EMPTY >

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

### Insert Response

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

<!ELEMENT row (value+)>

<!ELEMENT value (#PCDATA)*>

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

### PreInsert Response

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

<!ELEMENT row (value)*>

<!ELEMENT value (#PCDATA)*>

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

### Query Response

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

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

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

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

## Update Response

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

<!ELEMENT row (value+)>

<!ELEMENT value (#PCDATA)>

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

# Index