XML Reference: Siebel Enterprise Application Integration

Siebel Innovation Pack 2016

May 2016

ORACLE

Contents

Index

1 What's New in This Release

What's New in XML Reference: Siebel Enterprise Application Integration, Siebel Innovation Pack 2016

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

NOTE: Siebel Innovation Pack 2016 is a continuation of the Siebel 8.1/8.2 release.

What's New in XML Reference: Siebel Enterprise Application Integration, Siebel Innovation Pack 2015

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

NOTE: Siebel Innovation Pack 2015 is a continuation of the Siebel 8.1/8.2 release.

2 Overview of Support for XML in Siebel Business Applications

This chapter provides an overview of support for Extensible Markup Language (XML) in Siebel Business Applications. It includes the following topics:

About XML

XML is the industry standard for precisely representing data from virtually any source, stored in virtually any format. In appearance, it is similar to HTML, but while HTML explains a document in terms of how it should display data in a Web browser, XML *is* the data (or more precisely, the data from an application represented as XML).

This data can be from an application screen, sometimes called a *screen scraping,* it can be the output from a database, or it can be an application executed using processing instructions that run Oracle's Siebel eScript, for example.

There are also technologies that explain XML documents. These are known as *metadata* because the data within these documents is used to describe and format the information in an XML document. Examples of metadata documents include XSDs (XML Schema Definitions), DTDs (Document Type Definitions), and XDRs (XML Data Reduced), which are supported by Siebel Business Applications.

Siebel CRM Integration and XML

Siebel Business Applications support for XML allows you to communicate with any Siebel application or external application that can read and write XML (either arbitrary XML or Siebel XML, also known as the Siebel Message format).

XML documents are delivered directly to and from Siebel Business Applications, or through middleware using any of the supported transports: HTTP, IBM WebSphere MQ, File, and so on. XML communicated in this way can query the Siebel Database, upsert (update or insert) data, synchronize the two systems, delete data, or execute a workflow process.

Objects from various systems, such as Siebel business objects and Oracle application data, can be represented as Siebel integration objects.

Siebel CRM can also communicate bidirectionally with Web services using Simple Object Access Protocol (SOAP), and Representational State Transfer (REST) through Siebel Application Integration (SAI) for Oracle Fusion Middleware. For details, see *Integration Platform Technologies: Siebel Enterprise Application Integration* and *Siebel Application Integration for Oracle Fusion Middleware Guide*.

NOTE: If you do a minimal client installation, make sure you select the XML parser option; otherwise, you will encounter the following error when attempting to run any client process that uses the XML parser: *Unable to create the Business Service 'EAI XML Converter.'* The XML parser is included by default in the full installation.

XML Integration Objects

The Integration Object type of XML is available within Siebel Business Applications to represent externally defined XML documents, where the object's XML representation is compliant with the XSD or DTD supplied by your trading partner or external system. This type of integration object supports a representation of XML documents.

NOTE: Siebel XSD does not support the use of <import> and <include> elements and the <any> attribute. To implement the <import> or <include> functionality, place the schema definition into a single file.

Bidirectional Data Flow

Figure 1 shows the bidirectional progress of XML documents into and out of Siebel Business Applications.

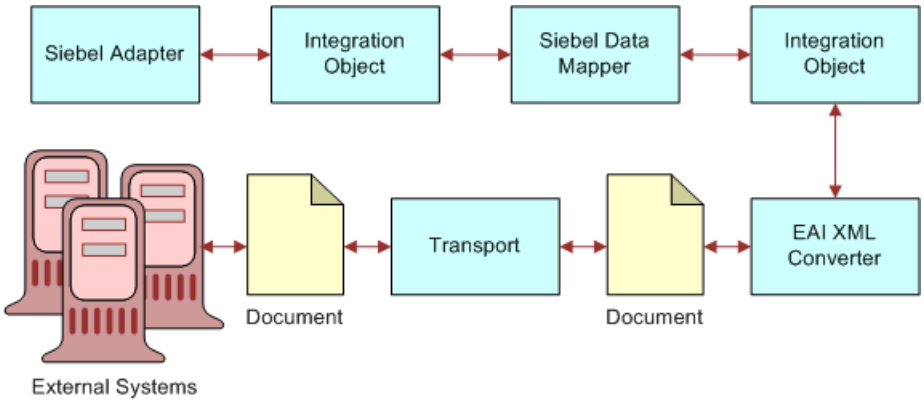

Figure 1. Document to Integration Object Flow

NOTE: For details on integration objects and Web services, see *Integration Platform Technologies: Siebel Enterprise Application Integration*. For an overview of Siebel EAI, see *Overview: Siebel Enterprise Application Integration*.

Metadata Support for XML

For sending and receiving information for Siebel Objects in an XML format between Oracle and external systems, Oracle supports the metadata representations for XML known as XSDs (XML Schema Definitions), DTDs (Document Type Definitions), and XDRs (XML Data Reduced, a Microsoft specification). Support for XSDs and DTDs gives you a way to communicate with external systems using externally defined XML documents, instead of having to use the Siebel XSD and DTD format.

The Siebel application includes a Schema Generator wizard to assist in the creation of XML Integration Objects, using an externally defined XSD or DTD. The XSD and DTD are used to map data between the Siebel application and an external integration object, and to transform data, as needed. These tasks are conducted using the Siebel Data Mapper.

Special Characters in XML Documents

Special characters should be represented in accordance with XML standards for those characters in order for them to be correctly interpreted within Siebel Business Applications. Also, specify the character set you are using if it is not UTF-8 (the default).

NOTE: To edit an XML document including binary or encoded data, use editors such as Microsoft Notepad or Word that do not convert the data upon saving the file.

Special (Escape) Characters

The EAI XML Converter can handle special characters for inbound and outbound XML, as shown in Table 1. Non-Siebel XML should already handle special characters before integrating into the Siebel application. Special characters are indicated by enclosing the text for the character between an ampersand (&) and a semicolon (;). Also, if the XML is passed in a URL, then URL encoding of special characters is required as shown in Table 1.

Table 1. XML Escape Characters (Character Entities)

Character	Entity	URL Encoded
<	<	%26lt%3B
>	>	%26gt%3B
&	&	%26amp%3B
“	"	%26quot%3B
'	'	%26apos%3B
Unicode Character (Decimal)		%26%2309%3B
Unicode Character (Hex)	°	%26%23x00B0%3B
Date	Must follow the ISO 8601 format	Not applicable

Declaring the Character Set in Use

You must include the following parameter in the XML version declaration of your XML, XSD, or DTD document to declare the character set in use, if it is not the default of UTF-8:

```
<?xml version="1.0" encoding="US-ASCII"?>
```

Supported character sets include but are not limited to ASCII, UTF-8, UTF-16 (Big or Small Endian), UCS4 (Big or Small Endian), EBCDIC code pages IBM037 and IBM1140 encodings, ISO-8859-1, and Windows-1252. This means that the XML parser can parse input XML files in these encodings.

The following encodings can be used in the XML declaration:

- US-ASCII
- UTF-8
- ISO-10646-UCS-4
- ebcdic-cp-us
- ibm1140
- ISO-8859-1
- windows-1252

The character set declaration encoding must appear after the version declaration. For example: `<?xml version="1.0" encoding="US-ASCII"?>`

The output can be in one of the following XML encodings:

- UTF-8
- UTF-16
- Local Code Page

3 XML Representation of Property Sets

This chapter discusses the XML representation of property sets and the mapping between property sets and XML. It also discusses the elements and attributes naming conversion performed by the XML Converter. It includes the following topics:

Mapping Between Property Sets and XML

An arbitrary property set hierarchy can be serialized to XML and an XML document can be converted to a property set hierarchy using the XML Converter business service. This service is used by the Business Service Simulator screen to save property set inputs and outputs to a file from eScript.

Each part of a property set object has a corresponding XML construct. Table 2 shows the mappings between parts of a property set hierarchy and their XML representation.

Table 2. Property Set to XML Mappings

Property Set Component	XML Representation
PropertySet	Element
PropertySet Type	Element name (if Type is not specified, then the element name is set to PropertySet)
PropertySet Value	Element Character Data
Property name	Attribute name
Property value	Attribute value
Child Property Set	Child element

Element and Attribute Naming

The property set Type (which maps to an XML element name) and the names of individual properties (which map to XML attribute names) do not necessarily follow the XML naming rules. For example, a name can include characters such as a space, a quote, a colon, a left parenthesis, or a right parenthesis that are not allowed in XML element or XML attribute names. As a result, you must perform some conversion to generate a valid XML document.

When creating an XML document from a property set hierarchy, the XML Converter will make sure that legal XML names are generated. There are two different approaches provided to handle name translation. The approach is determined by the *EscapeNames* user property on the XML Converter service. This user property can be either True or False.

- **True.** If EscapeNames is True, instances of illegal characters are converted to an escape sequence that uses only legal characters. For example, a space is converted to the characters *_spc*. When an XML document is parsed to a property set hierarchy, the escape sequences are converted back to the original characters. For example, the name Account (SSE) becomes Account_spc_lprSSE_rpr.

  Table 3 shows the escape sequences that are used by the XML Converter.

Table 3. XML Converter Escape Sequences

Character in Property Set	Description	Generated Escape Sequence
	Space	_spc
_	Underscore	_und
“	Double Quote	_dqt
‘	Single Quote	_sqt
:	Colon	_cln
;	Semicolon	_scn
(	Left Parenthesis	_lpr
)	Right Parenthesis	_rpr
&	Ampersand	_amp
,	Comma	_cma
#	Pound symbol	_pnd
/	(Forward) slash	_slh
?	Question Mark	_qst
<	Less Than	_lst
>	Greater Than	_grt
Illegal characters	Other illegal characters not listed in this table	_<Unicode character code>

- **False.** If EscapeNames is False, the XML Converter removes illegal characters. These characters include the space ( ), double quote ("), single quote('), semicolon (;), left parenthesis ((), right parenthesis ()), and ampersand (&). For example, the XML Converter changes the name Account (SSE) to AccountSSE.

  NOTE: These conversions are not reversible: the original names cannot be obtained from the XML names.

If a property set instance does not have a value for its Type member variable, the XML Converter uses the name PropertySet for the corresponding element's name.

Property Set Examples and Their XML Representation

The following is examples of different types of property sets that are available and their XML representation:

An Arbitrary Property Set

```
<?Siebel-Property-Set> <PropertySet> <Person> Jack </Person> </PropertySet>
```

A Siebel Message

```
<?Siebel-Property-Set EscapeNames="true"><PropertySet><SiebelMessage MessageID="1-
111" IntObjectFormat="Siebel Hierarchical" MessageType="Integration Object"
IntObjName="Sample Account"><ListOfSample_spcAccount>...</
ListOfSample_spcAccount></SiebelMessage></PropertySet>
```

An XML Hierarchy

```
<?Siebel-Property-Set><PropertySet><_XMLHierarchy><Account><Contact>...</
Contact></Account><_XMLHierarchy></PropertySet>
```

Figure 2 illustrates an example property set hierarchy and the XML that would be generated for each component of the hierarchy. The XML was generated with the EscapeNames user property set to True.

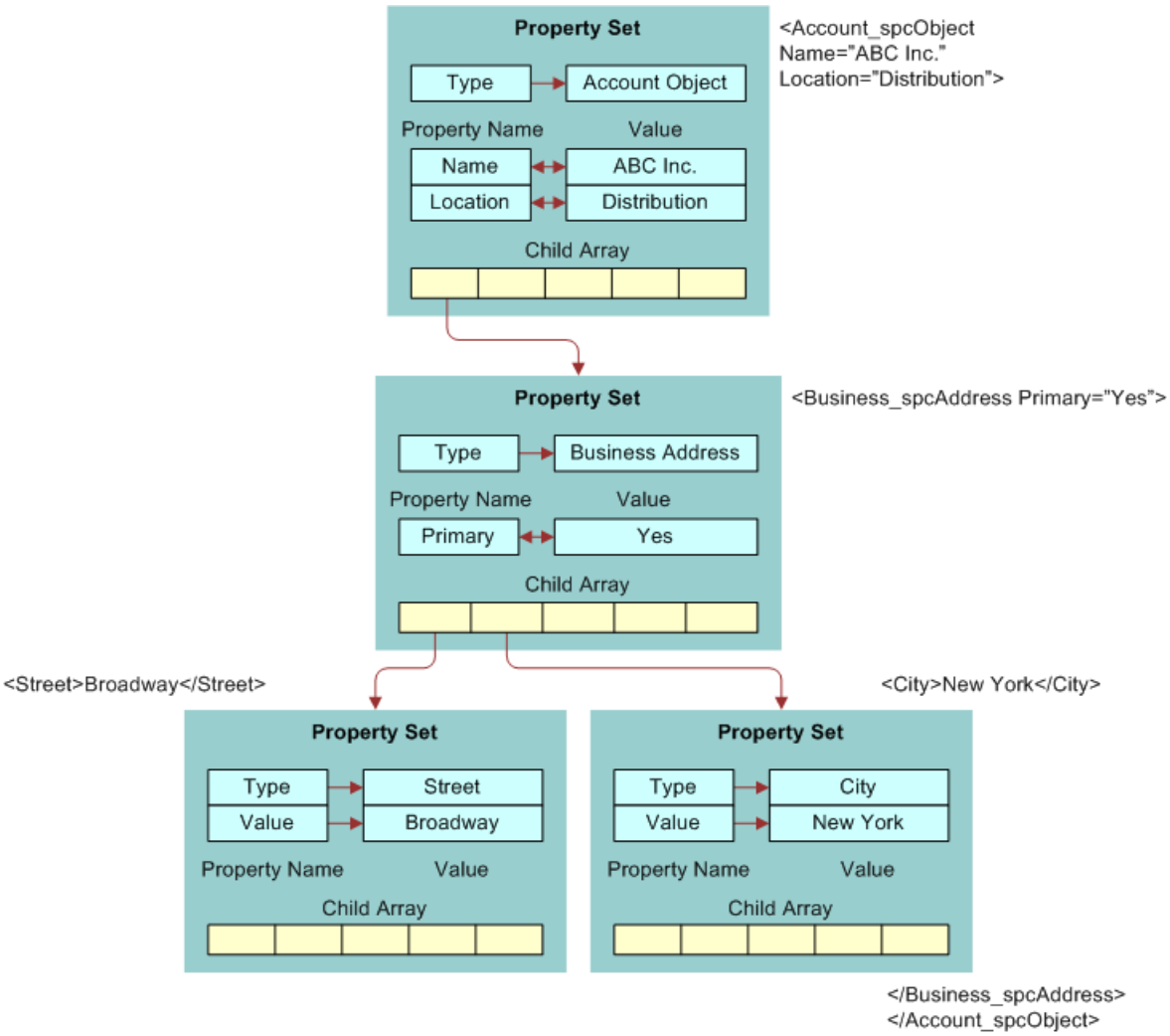

Figure 2. Property Set and XML with EscapeNames Set to True

Properly Formatted 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.

To benefit from using the XML Converter, be sure that any code you use, such as eScript or Siebel VB, correctly represents property sets within Siebel Business Applications for the XML Converter Business Service. This includes necessary arguments and values. An example of such code is:

```
Set Inputs = TheApplication.NewPropertySet
```

```
REM Fill in Siebel Message Header
Inputs.SetType "SiebelMessage"
Inputs.SetProperty "MessageId", ""
Inputs.SetProperty "MessageType", "Integration Object"
Inputs.SetProperty "IntObjectName", "Sample Account"

Set svc = theApplication.GetService("EAI XML Converter")
Set XMLInputs = theApplication.NewPropertySet
Set XMLOutputs = theApplication.NewPropertySet

XMLInputs.AddChild Inputs

svc.InvokeMethod "PropSetToXML", XMLInputs, XMLOutputs
```

4 XML Representation of Siebel Integration Object Instances

This chapter describes the XML representation of Siebel integration object instances. It includes the following topics:

About Representing Siebel Integration Object Instances as XML Documents

You can represent any integration object instance in Siebel Business Applications as an XML document (or created from a properly formatted XML document). This makes it convenient to save an object to a file for viewing or to send it over a transport, such as HTTP or IBM WebSphere MQ. You can control the format of the XML document through the integration object definition in the Siebel Repository. You can use the EAI XML Converter business service to perform translations between integration object instances and the corresponding XML representation.

Integration Objects

Integration objects are logical representations of Siebel business objects or external application data, such as externally defined XML documents. An integration object is metadata stored in the Siebel Repository. One integration object can be mapped to another integration object. Instances of integration objects are used in integration processes for data exchange. For more information on integration objects, see *Integration Platform Technologies: Siebel Enterprise Application Integration*.

Integration objects are made up of three distinct data sections: the canonical, the external, and the XML, as shown in Figure 3.

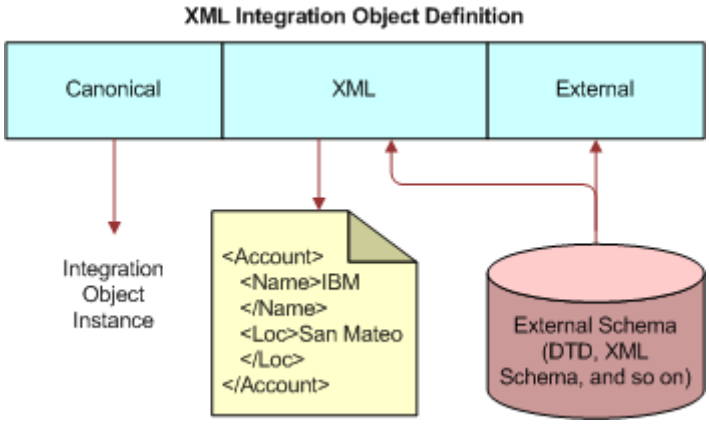

Figure 3. XML Integration Object Definition

The integration object schema in the Siebel Repository is composed of the three data sections shown in Table 4.

Table 4. Integration Object Data Type

Name	Purpose
Canonical section	Stores information about an object in a common representation. The names used for objects, components, and fields are the names that the designer wishes to be visible. The data types are the Siebel business component field types that are used by the Object Manager.
External section	Stores information about how the object, component, or field is represented in the external system. For integration objects based on business objects, this can include the business object names, component names, and field names and data types.
XML section	Stores the mapping between an integration object definition and its XML representation. This allows any integration object to be represented as XML.

Elements and Attributes

An XML document consists of one or more *elements*. An element consists of a start tag and an end tag that enclose character data, nested elements, or both. For example, here is a simple element called *Element1*, with two tags containing character data:

```
<Element1>
This is character data.
</Element1>
```

The next example shows an element nested within another element. Parent-child relationships are frequently represented using nested elements.

```
<Element1>
  <NestedElement>
  data
  </NestedElement>
</Element1>
```

Elements can have attributes that refine or modify the element's default attributes. An attribute is a key and value pair of strings, contained within the start tag of an element. In the following example, *status* is an attribute that is assigned the value *test*. Attributes are frequently used to specify metadata about an element.

```
<Element1 status="test">
This is character data.
</Element1>
```

In the Siebel representation, objects and components are represented by XML elements. A set of integration object *instances* of a given type are nested within the *object* element for that type.

An element represents each *component*. Child components are nested within their parent's elements. Fields can be either elements nested within their containing component element or attributes of the component element. You can set the XML Style attribute of the integration component field definition to specify which style represents a given field.

How XML Names Are Derived from Integration Objects

When Siebel Tools generates the XML representation of your integration object, it derives the XML element and attribute names from the Siebel Repository names of the integration object, its components, and fields. However, Siebel Repository names can include characters not permitted in an XML name, such as blank spaces. Thus, some translation must be performed to make sure a valid XML name is derived from such a repository name. In addition, XML element names must be unique in the document in which they are defined. This can cause a parsing problem if two integration components have fields with the same name.

To handle these issues, Siebel Tools stores a separate name in the XML Tag attribute of the integration object, component, and field. When you create an integration object using a wizard, the *XML Tag* attribute is initialized to the value of the *Name* column, with any illegal characters removed from the name. In addition, Siebel Tools might add a number to the tag name if the same name is already in use by a different object, component, or field. You can change the XML names after the integration object has been created, if necessary.

Elements Within a Siebel Integration Object Document

An integration object can be textually represented as an XML document. In order to exchange data using the Siebel integration object document, you must have an understanding of its XML structure, including elements and attributes. The document can include up to five different types of elements:

SiebelMessage Element

When integration object documents are sent to an external system, they might be encapsulated within a *SiebelMessage* element. This element identifies the document as a Siebel message and indicates that the document includes integration object instances. It can also provide metadata, such as the integration object type and a message ID.

NOTE: The SiebelMessage element is optional. The presence of this element is determined at run time through arguments to the EAI XML Converter Business Service.

Since the Object List element is optional, SiebelMessage can include a Root component element to allow cases when the Object List element is left blank (omitted). For details on Object List element, see "Object List Element" on page 23.

Attributes

The SiebelMessage element can contain a number of attributes, which are known as the Message Header attributes. In addition, you can add arbitrary attributes to the SiebelMessage element. An XSD or DTD for the document can be dynamically generated inline to include all present attributes. The following standard attributes have well-defined meanings.

IntObjectName

The name of the integration object type contained within the message. If the message is an integration object message, you must specify this property.

MessageId

A unique ID for a given message as it flows through a connector. This is an optional field that might be useful for tracking message processing.

Child Elements

For integration object messages, the SiebelMessage element includes exactly one *object list element*. Since only one object list element is permitted in each XML document, only one *integration object type* can be represented in a given document.

Object List Element

The *object list element* is a container for the integration object instances. The XML Tag attribute value that you specify in the integration object definition becomes the name of this element. By default, an integration object wizard generates an XML Tag value of ListOf*Name*, where *Name* is the name of the integration object, with any illegal XML characters removed—for example, spaces.

NOTE: The Object List element is optional. The XML element is not generated if the Object List element is blank (omitted) in the integration object definition.

Attributes

None.

Child Elements

The object list element can include one or more instances of the integration object's *root component element*. A root component element corresponds to one integration object instance.

Integration Component Elements

An *integration component element* corresponds to an *integration component type* in the repository definition.

Component parent-child relationships are represented by a structure in which the child components of a given component type are nested within a component container element. The *component container element* is, in turn, nested within the *parent component instance*.

Thus, all components within an integration object instance are indirectly nested within the root component. Only one instance of the root component is allowed for each object instance. The root component is nested within the object list element. The object list element permits multiple integration object instances of a given type within the XML document.

The field children of an integration component element can be either elements or attributes, depending on the XML Style setting for each field. The component container elements of a given component appear after the fields in the XML document.

In the following example, Contact child components are nested within the Account component instance:

```
<Account>
. . .
Account Field Elements
```

```
. . .
  <ListOfContacts>
    <Contact> . . . Contact 1 . . . </Contact>
    <Contact> . . . Contact 2 . . . </Contact>
  </ListOfContacts>
<Account>
```

Attributes

Any field that has an XML Style set to Attribute is an attribute of its component element. The name of the attribute is the same as the XML Tag of the field.

Child Elements

An integration component element can include integration field elements and component container elements. The field elements must appear before the component container elements. The name of a field element is determined by the value of its XML Tag attribute, as defined in Siebel Tools.

Component Container Elements

An *integration component container* encloses a list of *child component instances* of the same type. The integration component container organizes child component instances by type and permits the specification of empty containers—functionality needed by the EAI Siebel Adapter. All component types, except the root component, are enclosed within container elements.

By default, the name of a component container element is ListOf plus the element name of the component type it encloses. For example, the component container for Contact is ListOfContact. You can override the default name by specifying a name in the XML Container element field of the component's definition.

Another option is to leave the container element blank. In that case, the component element is the child of the parent component element.

Attributes

None.

Child Elements

Zero or more instances of the component element associated with the container.

Integration Field Elements

An *integration field element* includes the value of the specified field. It must appear in an instance of its parent integration object type. If a field element has no contents (signified by a start tag immediately followed by an end tag), it is interpreted to mean that the field's value should be set to empty. The same is true when a field's value is empty; the field element will have a start tag immediately followed by an end tag.

The order in which XML fields appear within their parent component element is determined by the Sequence field in the Tools definition of the field.

All fields are optional. If a field element is not present in a component element, the field is not created in the integration object instance.

Child Elements

Integration component fields have a property called XML Parent Element. If this property contains the name of another field, then that field (either as an attribute or as an element) appears as a child of its parent field's element.

Example XML Document

The following XML document represents an instance of the Sample Account integration object. This document includes one account instance: A. K. Parker Distribution. The instance has one business address and two contacts.

Note that the PhoneNumber field of the business address appears as an *attribute*. This means that the XML Style in the field's definition in Siebel Tools is set to the Attribute style. The rest of the fields are represented by XML elements.

```
<SiebelMessage MessageId=""
IntObjectName="Sample Account">
<ListofSampleAccount>
  <Account>
    <Name>A. K. Parker Distribution</Name>
    <Location>HQ-Distribution</Location>
    <Organization>Siebel Organization</Organization>
    <Division></Division>
    <CurrencyCode>USD</CurrencyCode>
    <Description></Description>
    <HomePage></HomePage>
    <ListOfBusinessAddress>
      <BusinessAddress PhoneNumber="6502955000">
      <City>Menlo Park</City>
      <Country>United States of America</Country>
      <FaxNumber></FaxNumber>
      <StreetAddress>1000 Industrial Way</StreetAddress>
      <Province></Province>
      <State>CA</State>
      <PostalCode>94025</PostalCode>
    </BusinessAddress>
```

```
    </ListOfBusinessAddress>
    <ListOfContact>
      <Contact>
        <FirstName>Stan</FirstName>
        <JobTitle>Senior Mgr of MIS</JobTitle>
        <LastName>Graner</LastName>
        <MiddleName></MiddleName>
        <Organization>Siebel Organization</Organization>
        <PersonalContact>N</PersonalContact>

  </Contact>
    <Contact>
      <FirstName>Susan</FirstName>
      <JobTitle>President and CEO</JobTitle>
      <LastName>Grant</LastName>
      <MiddleName></MiddleName>
      <Organization>Siebel Organization</Organization>
      <PersonalContact>N</PersonalContact>
    </Contact>
    <Contact>
</ListOfContact>
</Account>
</ListofSampleAccount>
</SiebelMessage>
```

XML Schema Definitions (XSDs)

The XML Schema Definition (XSD) language describes the content of an XML document. The definition can describe which elements are allowed and how many times the element can be seen. The schema can be used to generate an integration object through Siebel Tools. The feature is accessed through the Integration Object Builder.

Here is an example of an XSD for the Sample Account integration object as generated by Siebel Tools:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="http://
example.com/xsd/SampleAccount.xsd" xmlns:xsdLocal="http://example.com/xsd/
SampleAccount.xsd" >

<xsd:element name = "elem1" type ="xsd:string" minOccurs ="0" maxOccurs = "1"/>

<xsd:element name = "elem2" type ="xsd:string" minOccurs ="0" maxOccurs="unbounded"/
>

</xsd:schema>
```

NOTE: All Siebel data types except DTYPE_ATTACHMENT map to xsd:string. DTYPE_ATTACHMENT maps to xsd:base64Binary.

Document Type Definitions (DTDs)

The Document Type Definition (DTD) provides metadata describing the structure of an XML document. It can be used by validating XML parsers to make sure that a given document instance conforms to the expected structure, that is, the structure defined in the DTD.

You can generate the DTD for an integration object by using the Generate Schema feature in Siebel Tools. The feature is activated by clicking the Generate Schema button in Siebel Tools after selecting a given integration object definition.

NOTE: Attachment attributes are not supported in DTDs because they are not part of the integration object definition and only appear at runtime.

The SiebelMessage element is optional. It can be omitted by selecting the No Envelope option in the Generate XML Schema wizard.

The DTD for the message header is generated in the actual XML document at run-time. The generation of this inline DTD and a reference to the external portion is enabled through the GenerateDTD parameter of the EAI XML Converter.

Here is an example of a DTD for the Sample Account integration object as generated by Siebel Tools:

```
<!-- Siebel DTD Generation -->
<!-- Shared Element List. These elements are guaranteed -->
<!-- to have the same datatype, length, precision, and scale.-->
<!ELEMENT Name (#PCDATA) >
<!ELEMENT Location (#PCDATA) >
<!ELEMENT Division (#PCDATA) >
<!ELEMENT Description (#PCDATA) >
<!ELEMENT CurrencyCode (#PCDATA) >
<!ELEMENT StreetAddress (#PCDATA) >
<!ELEMENT State (#PCDATA) >
<!ELEMENT PostalCode (#PCDATA) >
<!ELEMENT Country (#PCDATA) >
<!ELEMENT City (#PCDATA) >
<!ELEMENT Organization (#PCDATA) >
<!ELEMENT ListofSampleAccount (Account+) >
<!ELEMENT Account (Name?,
      Location?,
      Organization?,
      Division?,
      CurrencyCode?,
      Description?,
      HomePage?,
      LineofBusiness?, BusinessAddress?, Contact?)>
<!ELEMENT HomePage (#PCDATA) >
<!ELEMENT LineofBusiness (#PCDATA) >
<!ELEMENT BusinessAddress (BusinessAddress*) >
<!ELEMENT BusinessAddress (City?,
      Country?,
      FaxNumber?,
      StreetAddress?,
      Province?,
      State?,
```

```
      PostalCode?)>
<!ATTLIST BusinessAddress PhoneNumber CDATA #IMPLIED >
<!ELEMENT FaxNumber (#PCDATA) >
<!ELEMENT Province (#PCDATA) >
<!ELEMENT Contact (Contact*) >
<!ELEMENT Contact (CellularPhone?,
      FirstName?,
      HomePhone?,
      JobTitle?,
      LastName?,
      MiddleName?,
      Organization?,
      PersonalContact?,
      Account?,
      AccountLocation?)>
<!ELEMENT CellularPhone (#PCDATA) >
<!ELEMENT FirstName (#PCDATA) >
<!ELEMENT HomePhone (#PCDATA) >
<!ELEMENT JobTitle (#PCDATA) >
<!ELEMENT LastName (#PCDATA) >
<!ELEMENT MiddleName (#PCDATA) >
<!ELEMENT PersonalContact (#PCDATA) >
<!ELEMENT Account (#PCDATA) >
<!ELEMENT AccountLocation (#PCDATA) >
```

NOTE: All fields are optional, but if they are present, then they must appear in the correct order. The definition of a field appears only once at the beginning of the DTD, even if its XML tag appears in multiple components. When creating XML tag names for fields, the wizard only reuses a field name if all instances have the same data type, length, precision, and scale.

5 XML Integration Objects and the XSD Wizard

This chapter discusses the XSD wizard, the supported XSD elements and attributes, and the structure of the XSD XML integration object, such as user properties. It includes the following topics:

Creating XML Integration Objects with the XSD Wizard

Siebel EAI provides two different wizards to create XML integration objects. An XML integration object is essentially an integration object with a base object type of XML. This wizard parses the XML Schema Definition (XSD) file to create an XML integration object.

To create an integration object

1. Launch Siebel Tools.
2. Select File, then New Object.
3. In the New Object Wizards window, select the EAI tab.
4. Double-click the Integration Object icon.
5. Complete the Integration Object Builder initial page:
   a. Select the project from the first drop-down list.
   b. Select EAI XSD Wizard as the Business Service.
   c. Navigate to the location of the XSD or XML file that you want to use as the basis of the XSD and click Next.

      NOTE: The Simplify Integration Object Hierarchy option creates a simpler and flatter internal representation of the XML integration object; however, this does not change the external representation. Having a simpler internal representation makes declarative data mapping easier.
6. Select the source object, give it a unique name, and then click Next.
7. Click on the plus sign to expand the list and select or clear the fields you need from the component.
8. Click Next to get to the final page to review the messages generated during the process and take necessary action as required.

9 Click Finish to complete the process.

The integration object is displayed in the Integration Objects list.

NOTE: You must review the integration objects and the integration components created by the Wizard and complete their definitions based on your requirements.

Selecting the Source Object in the XSD Wizard

Each XML document has exactly one root or document element. The root element corresponds to the integration object. However, because an XSD or DTD file can be used by a vendor to specify the XML documents that it can generate, the root element cannot be inferred from the XSD or DTD file. For example, Ariba can generate XML for contracts, order requests, subscriptions, and so on. A single file describes the possible XML documents.

As a reference when determining the root element, use an XML document that best represents the XML documents you are integrating. The root element is the root of the XML hierarchy tree. No part of the root element appears within the content of any other element. For all other elements, the <Start></Start> tag appears within the content of another element.

To view any XML hierarchy, with collapsible and expandable elements, use an XML editor, an XML reader, or an XML-capable browser such as Microsoft Internet Explorer.

Supported XSD Elements and Attributes

Not all XSD schema elements and attributes are supported by Siebel Business Applications. Table 5 on page 31 and Table 6 on page 34 list all the XSD elements and attributes with Siebel CRM support levels for them. The following terminology is used in these tables:

- **Ignored.** This level of support means that processing will continue, and an error is not generated. However, the information given for the specified element or attribute is ignored.
- **Mapped.** This level of support means that the information specified in a given element or attribute is used in the integration object representation.
- **Not mapped.** This level of support means that the given element or attribute information is not used. However, children of the element will be processed.

NOTE: The Siebel application does not perform any formatting or processing for any of the schema types. All the scalar types such as string, ID, or integer are treated as strings. When converted to an integration object and integration component field, DataType is set to DTYPE_TEXT.

Table 5 lists XSD schema elements and the level of support for them in Siebel Business Applications.

Table 5. XSD Schema Elements and Siebel CRM Support Level

Element	Siebel CRM Support Level	Details
all	Not mapped. Treated as sequence.	Not applicable
annotation	Mapped	Mapped as a parent's comment property. Children can be mapped only if parent of annotation is mapped to a component or field.
any	Mapped	Mapped as a XML Hierarchy if namespace attribute cannot be resolved to a schema import definition. Otherwise, all global elements logically replace the any element that are then mapped to an integration object using rules for elements. Acts as a placeholder for any element. For more information about this element, see *Integration Platform Technologies: Siebel Enterprise Application Integration*.
anyAttribute	Mapped	Same as the any element. Act as a placeholder for any attribute. For more information about this element, see *Integration Platform Technologies: Siebel Enterprise Application Integration*.
appinfo	Ignored	Not applicable
attribute	Mapped	Mapped as a field. Storing type information is useful when generating schema either after importing one or manually creating one. Also, useful for type specific formatting, such as xsd:datetime.
attributeGroup	Mapped	Mapped as children attributes that are added as fields to the parent element's component.
choice	Not mapped. Treated as sequence.	Not applicable
complexContent	Mapped	Mapped to add properties and children to the parent element's component. Attributes can affect parent (complexType) and children when restriction and extension are processed.

Table 5. XSD Schema Elements and Siebel CRM Support Level

Element	Siebel CRM Support Level	Details
complexType	Mapped	Mapped if global complexType is starting point for integration object that maps to root component. Also mapped when XSDTypeName and XSDTypeNamespace user properties are set on the root or elements component.
documentation	Mapped	Mapped if Comment property is on a field, component, or object.
element	Mapped	Mapped as a component or field. If element is of simpleType and maxOccurs is at most 1, then map to field, otherwise map to component (complexType).
enumeration	Ignored	Not applicable
extension	Mapped	Mapped if merging base type and children into the parent. Extension element affects the parent for complexContent and simpleContent.
field	Ignored	Not applicable
group	Mapped	Mapped if adding children to the parent element's component.
import	Mapped	Preprocessed to receive the additional schema. Resolve a schemaLocation reference by URI or Local (File). Whatever is defined in imported schema will belong to a different namespace.
include	Mapped	Preprocessed to receive the additional schema. Resolve a schemaLocation reference by URL or Local (File). Whatever is defined in imported schema can belong to the same namespace.
key	Ignored	Defines a unique key.
keyref	Ignored	Defines fields for key. Keyref refers to a key that must exist in the document.

Table 5. XSD Schema Elements and Siebel CRM Support Level

Element	Siebel CRM Support Level	Details
length	Mapped. Does not support lists.	Mapped for field external length and length. Fixed length of string-based content. Also might mean length of a list (number of items).
list	Ignored	Not applicable
maxLength	Mapped	Mapped for field length.
minExclusive, maxExclusive	Ignored	Not applicable
minInclusive, maxInclusive	Ignored	Not applicable
minLength	Not mapped	You can use minlength = 0 to indicate that a field can have zero characters, that is, it is optional. You must manually edit the XSD to specify the minLength value.
notation	Ignored	Not applicable
pattern	Ignored	Not applicable
redefine	Ignored	Not applicable
restriction	Mapped	Mapped when adding children to the parent component or field. Affects its parent: `complexContent`, `simpleContent`, `simpleType`. Remove the elements and attributes that are not specified as the restriction ones. Validate that the elements and attributes used in the restriction are present in the base type.
schema	Mapped	Namespace information used for object, component, and field.
selector	Ignored	Not applicable
sequence	Not mapped	Not applicable
simpleContent	Mapped	Mapped when adding properties and children to the parent element's component.
simpleType	Mapped	XSDTypeName and XSDTypeNamespace user properties on parent element's field or component, or attribute's field.
union	Ignored	Not applicable
unique	Ignored	Not applicable

Table 6 lists XSD schema attributes and the level of support for them in Siebel Business Applications.

Table 6. XSD Schema Attributes and Siebel CRM Support Level

Attribute	Siebel CRM Support Level	Details
abstract	Ignored	Not applicable
attributeFormDefault	Ignored	Not applicable
base	Mapped	Mapped if base type is used to create component or field.
block	Ignored	Not applicable
blockDefault	Ignored	Not applicable
default: attribute	Mapped	Mapped to XML Literal value property only. Provides default value for an attribute when an attribute is missing.
default: element	Mapped	Mapped to XML Literal value property only. Provides default value for an element when an element is empty.
elementFormDefault	Ignored	Not applicable
final	Ignored	Not applicable
finalDefault	Ignored	Not applicable
fixed: attribute or element	Ignored	Not applicable
fixed: simpleType	Ignored	Not applicable
form	Ignored	Not applicable
itemType	Ignored	Not applicable
maxOccurs	Mapped	Maps to the cardinality upper bound on parent element's component. Maps to One or More (unbounded). If you want to preserve the maximum number of occurrences, then new column is needed.
memberTypes	Ignored	Not applicable
minOccurs	Mapped	Maps to the cardinality lower bound on parent element's component. Maps to Zero or One. If you want to preserve the minimum number of occurrences, then new column is needed.
mixed	Ignored	Not applicable

Table 6. XSD Schema Attributes and Siebel CRM Support Level

Attribute	Siebel CRM Support Level	Details
name	Mapped	Maps to the XML Tag of parent element (component, field) or attribute field or to the XSD Type Name on object, component, or field. Name of the schema component.
namespace: any, anyAttribute	Mapped	Namespace for the replacement elements and attributes.
namespace: import	Mapped	Maps to Namespace and XSDNamespace user property on components and fields that are being imported. Namespace for the imported elements and attributes.
nillable	Ignored	Not applicable
processContents	Ignored	Not applicable
public	Ignored	Not applicable
ref	Mapped	Mapped if metadata starting from global element or attribute that is being referred to is copied to the referring element (component, field) or attribute field.
schemaLocation	Mapped	Mapped if used for preprocessing of import or include
substitutionGroup	Ignored	Not applicable
targetNamespace	Mapped	Maps to XSD Type Namespace and XML Tag Namespace user properties on the integration object, imported component, or field. Schema targetNamespace to which all schema components definitions in a particular schema belong (children of *schema* element).
type	Mapped	Maps to XSDTypeName user property on element's component or field, or attribute's field.
use	Ignored	Not applicable
version	Ignored	Not applicable
whitespace	Ignored	Not applicable
xpath	Ignored	Not applicable

Structure of XSD XML Integration Objects

The structure of an XSD XML integration object is same as any other integration object. This topic discusses properties specific to XSD XML integration objects.

NOTE: For details on integration objects, see *Integration Platform Technologies: Siebel Enterprise Application Integration*.

XSD-Specific Integration Object Properties

Table 7 lists the integration object property that is used to represent XSD as an XML integration Object.

Table 7. Integration Object Properties for Representing XSD

Name	Project	Base Object Type	XML Tag
Name of the integration object. The value is provided through the wizard.	The project that the integration object is built in. The value is provided through the wizard.	XML	XML Tag used to represent the integration object.

XSD-Specific Integration Object User Properties

Table 8 lists integration object user properties for representing XSD as an XML integration object.

Table 8. Integration Object User Properties for Representing XSD

Name	Value	Description
XMLTagNamespace	*targetNamespace*	Namespace for the Element XML tags.
XSDTypeName	*Name of the root complexType*	Name of the root complexType used to create the integration object. This is only used through WSDL Import.
XSDTypeNamespace	*targetNamespace*	The namespace URI of the root complex type. This is only used through WSDL Import.

XSD-Specific Integration Component Properties

Table 9 lists the integration component property for representing XSD as an XML integration component.

Table 9. Integration Component Properties for Representing XSD

External Name Context	Name	External Name	External Sequence	Cardinality	XML Tag	XML Sequence
XPath to the schema component starting with the global element	XML Tag plus a sequence number to make component name unique within the integration object	Element name	XML Sequence	Based on minOccurs or maxOccurs	Element Name	Sequence within parent element

XSD-Specific Integration Component User Properties

Table 10 lists integration component user properties for representing XSD as an XML integration component.

Table 10. Integration Component User Properties for Representing XSD

Name	Value	Description
XMLTagNamespace	*targetNamespace*	Namespace for the Element XML tags.
XSDTypeName	*Component element type attribute*	Type of the element. For instance, for element type="xyz", XSDTypeName=xyz.
XSDTypeNamespace	*NamespaceURI for element type*	Namespace for the element type. This is the Namespace URI for the element's type name.

XSD-Specific Integration Component Field Properties

Table 11 lists the integration component field property for representing XSD as an XML integration component.

Table 11. Integration Component Field Properties for Representing XSD

Name	Data Type	Length	External Name	External Length	XML Literal Value
XML Tag	DTYPE_TEXT	maxLength or length	Attribute or element name	Length	fixed or default

XSD-Specific Integration Component Field User Properties

Table 12 lists integration component field user properties for representing XSD as an XML integration component.

Table 12. Integration Component Field User Properties for Representing XSD

Name	Value	Description
XMLTagNamespace	*targetNamespace*	Namespace for element or attribute XML tags.
XSDTypeName	*Field element or attribute XML Schema type name*	Type of the element or attribute. For instance, for element type =“xyz”, XSDTypeName=xyz.
XSDTypeNamespace	*NamespaceURI for element or attribute type*	Namespace for the element or attribute type. In effect, this is the Namespace URI for the element's or attribute’s type name.

6 XML Integration Objects and the DTD Wizard

This chapter discusses the DTD wizard and how it creates XML integration objects. It includes the following topics:

Creating XML Integration Objects with the DTD Wizard

Siebel EAI provides two different wizards to create XML integration objects. An XML integration object is essentially an integration object with a base object type of XML. This wizard parses an external Document Type Definition (DTD) file to create an XML integration object.

To create an integration object

1. Select File, then New Object.
2. Select the EAI tab.
3. Double-click the Integration Object icon.
4. Complete the Integration Object Builder initial page:
   a. Select the project from the first drop-down list.
   b. Select EAI DTD Wizard as the Business Service.
   c. Navigate to the path to the location of the DTD or XML file that you want to use as the basis of the DTD and click Next.

      NOTE: The Simplify Integration Object Hierarchy option creates a simpler and flatter internal representation of the XML integration object. Please note that this does not change the external representation. Having a simpler internal representation makes declarative data mapping easier.
5. Select the source object and give it a unique name, and then click Next.
6. Click on the plus sign to expand the list and select or clear the fields based on your business requirements.
7. Click Next to go to the final page to review messages generated during this process and take necessary action.

8 Click Finish to complete the process.

The integration object is displayed in the Integration Objects list.

NOTE: You must review the integration objects and the integration components created by the Wizard and complete their definitions based on your requirements.

Selecting the Source Object in the DTD Wizard

Each XML document has exactly one root or document element. The root element corresponds to the integration object. However, because an XSD or DTD file can be used by a vendor to specify the XML documents that it can generate, the root element cannot be inferred from the XSD or DTD file. For example, Ariba can generate XML for contracts, order requests, and subscriptions. A single file describes the possible XML documents.

As a reference when determining the root element, use an XML document that best represents the XML documents you are integrating. The root element is the root of the XML hierarchy tree. No part of the root element appears within the content of any other element. For all other elements, the <Start></Start> tag appears within the content of another element.

To view any XML hierarchy, with collapsible and expandable elements, use an XML editor, an XML reader, or an XML-capable browser such as Microsoft Internet Explorer.

How the DTD Wizard Creates XML Integration Objects

XML integration objects consist of the following:

- "Elements" on page 40
- "Attributes" on page 41
- "Element's #PCDATA" on page 41
- "Names" on page 41
- "Hierarchy" on page 42
- "Connectors" on page 42
- "Cardinality" on page 42

CAUTION: The DTD Wizard removes recursion by breaking loops. Repeating entities in XML at run time are not supported.

Elements

Generally, XML elements map to components within integration objects. However, in many cases the component is so simple that it is a performance optimization to map these elements into component fields of the parent element rather than as child components.

Elements are expressed this way (within brackets and starting with an exclamation point):

```
<!ELEMENT car (year, model, color+)>
```

An element can be mapped to a component field when the following three properties are satisfied:

- The component field must match an element within the DTD.
- The component field must match the cardinality of the element in the DTD; in other words, if the DTD specifies only one instance of this element type is valid, all subsequent appearances of this element type are illegal.
- The element must appear *within* the root element; any element appearing outside of the root is illegal.

When an element is mapped to component field, the component field has the property XML Style set to Element.

Attributes

Attributes include additional information related to an element, can be either required or implied (optional), and might have a default value. For example, an element might be a car with soundsystem, transmission, and doors as attributes. Soundsystem can be any text and is required; transmission is required because there is a default listed; other is optional.This would be expressed this way:

```
<!ELEMENT car>
<!ATTLLST car
        soundsystem      CDATA        #REQUIRED
        transmission     (automatic | manual | 5-speed-manual) "automatic"
        other            CDATA        #IMPLIED>
```

Attributes are always mapped to component fields and are related directly to elements. The component field within Siebel application has the XML Style property set to Attribute.

Element's #PCDATA

If the element is mapped to an integration component, then its #PCDATA is mapped to a component field <!Element> #PCDATA. If the element is mapped to a field, then the #PCDATA is mapped to the value of the field.

Names

Name is the name of the component or the field of the integration object. Because these names have to be unique within an integration object, the names might have suffixes attached to make them unique.

- Property *External Name* is the name of the attribute or the element in the external system, such as CustName.
- Property *XML Tag* is the name of the tag in the XML, such as <customer>.

Hierarchy

The parent components of integration components in an integration object correspond to their parents in XML. For integration component fields, if the property *XML Parent Field* is set, then the field in the same component with its *Name* value equal to the *XML Parent Field* corresponds to the parent in the XML. This happens because elements can be mapped to fields of integration components.

For integration component fields, if the property *XML Parent Field* is not set, then the parent component corresponds to the parent in the XML.

Connectors

Connectors specify the order of elements and *either/or* relationships, if one exists, as shown in Table 13.

Table 13. Connectors

Connector	Explanation	Example
,	followed by	(a,b)
|	one or the other	(a | b)

CAUTION: The Siebel DTD wizard does not support “one or the other” (|) relationships expressed in DTDs. “One or the other” (|) will be treated the same as “followed by” (,).

Cardinality

As shown in Table 14, the DTD syntax allows you to specify a cardinality (the number of times an element can appear within an XML document) for elements using the following modifiers: question mark (?), plus sign (+), and asterisk (*), or none. Elements with a cardinality, or occurrence, specified in a DTD map only to Integration Components. The Cardinality property in the Integration Component within Siebel maps to the specified *cardinality* information in the DTD.

Table 14. Rules for Mapping for Cardinality Within a DTD

DTD Element Occurrence Operator	Description	Integration Component Cardinality Property	Description
None	Appears once	Not applicable	Not applicable
?	Appears 0 or once	Zero or One	Appears 0 or once
+	Appears one or more times	One or More	Appears one or more times
*	Can appear 0 or more times	Zero or More	Can appear 0 or more times
No modifier	Appears once	One	Appears once

The specification for DTDs supports using parentheses to express complex hierarchical structures. For example:

```
<!ELEMENT rating ((tutorial | reference)*, overall)+ >
```

The DTD Wizard uses the operator (?, *, +, or “none”) closest to the child element as that child element’s cardinality. In addition, the DTD Wizard will ignore such grouping by parentheses as illustrated in the preceding example.

7 Siebel XML Converters

This chapter provides detailed information about the various Siebel XML converters. It includes the following topics:

About Siebel XML Converters

Siebel EAI includes four XML converter business services:

- "EAI XML Converter" on page 46
- "XML Hierarchy Converter" on page 51
- "EAI Integration Object to XML Hierarchy Converter" on page 57
- "XML Converter" on page 60

NOTE: XML converters might add unexpected carriage returns throughout the output document, for readability reasons. These characters are not significant and can be removed if the receiving application does not expect them and produces a parsing error. You can use eScript or Siebel VB to remove them.

Table 36 on page 62 outlines the differences among these converters. Using these converters, Siebel EAI supports three types of standard XML integrations:

- **XML integration using Siebel XML.** This integration uses XML that conforms to the XML Schema Definition (XSD), Document Type Definition (DTD), or schema generated from any Siebel integration object. Siebel XML is generated by the external application or by a third-party product. This type of integration uses the EAI XML Converter business service.
- **XML integration without using integration objects.** For this integration, any necessary data mapping and data transformation must be handled using custom eScripts. This type of integration uses the XML Hierarchy Converter business service.

- **XML integration using XML integration objects.** With this integration, XML integration objects are mapped to Siebel integration objects using Siebel Data Mapper and are based on external XSDs or DTDs. XML integration objects are used to map data between the external application and Siebel Business Applications. This type of integration uses the EAI XML Converter business service.

NOTE: These converters do not support Shift-JIS page code on UNIX platforms.

You can specify most parameters for the XML Converters as either business service method arguments or as user properties on the business service. If a business service method argument and a user property have the same name, the business service method argument always takes precedence over the user property.

NOTE: There are also two associated business services for XML that combine XML Converters with file reading and writing, which are useful for testing and debugging during the development phase. These are the EAI XML Read from File business service and the EAI XML Write to File business service.

EAI XML Converter

The EAI XML Converter uses integration object definitions to determine the XML representation for data. It converts the data between an integration object hierarchy and an XML document. Figure 4 shows the translation of an XML document into an integration object property set in Siebel application and back again. The integration object property set of type Siebel Message will appear as a child of the Service Method Arguments property set.

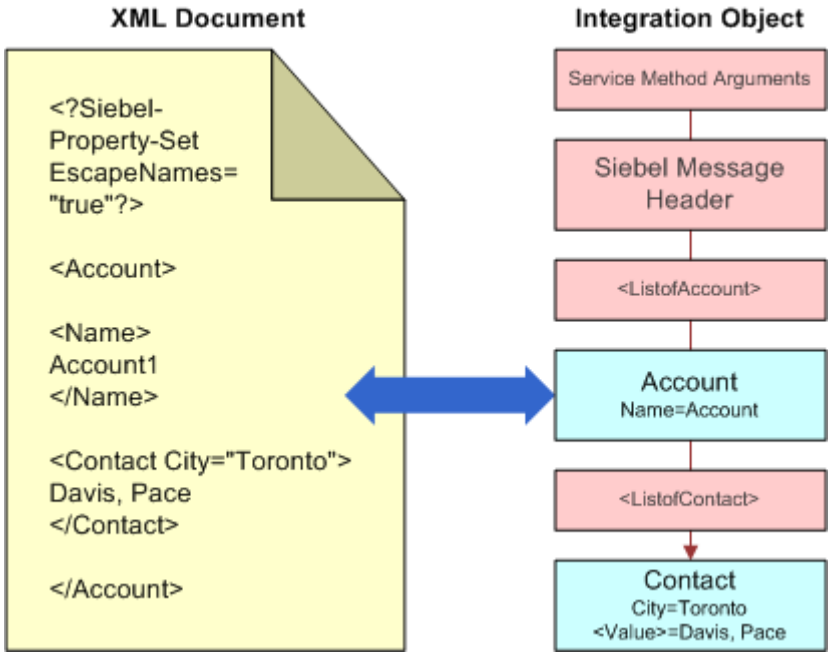

Figure 4. XML Document to Integration Object

The following topics are also described here:

EAI XML Converter Parameters

You can control the location of where you want the temporary EAI XML Converter file generation to occur as well as the threshold by setting two server component parameters as described in Table 15. For more information about setting these and other server component parameters, see *Siebel System Administration Guide*.

Table 15. EAI XML Converter Parameters

Server Component Parameter Name	Server Component Type	Description
XMLTempFilePath	EAIXMLConvSubsys	Use this parameter to specify the location where you want the temporary XML conversion files generated when EAI XML Converter response size is greater then the size specified in the XMLTempFileSize parameter. By default, if the response size is greater than 50 kb, then generation occurs in the *SIEBSRVR_ROOT*\temp directory.
XMLTempFileSize	EAISubSys	Use this parameter to specify the threshold size for EAI XML Converter method responses. For more information about these methods, see "Siebel XML Converter Business Service Comparison" on page 62.

EAI XML Converter Business Service Methods

There are two methods for the EAI XML Converter: Integration Object Hierarchy to XML Document and XML Document to Integration Object Hierarchy, as described in Table 16. The arguments for each method appear in Table 17, Table 18 on page 49, Table 19 on page 49, and Table 20 on page 51.

Table 16. EAI XML Converter Methods

Display Name	Name	Description
Integration Object Hierarchy to XML Document	IntObjHierToXMLDoc	Converts an integration object hierarchy into an XML document.
XML Document to Integration Object Hierarchy	XMLDocToIntObjHier	Converts an XML document into an integration object hierarchy.

Integration Object Hierarchy to XML Document Method Arguments

Table 17 describes the input arguments for the Integration Object Hierarchy to XML Document method of the EAI XML Converter.

Table 17. Integration Object Hierarchy to XML Document Method Input Arguments

Display Name	Name	Data Type	Description
Siebel Message	SiebelMessage	Hierarchy	The Integration Object Hierarchy to be converted to XML.
XML Character Encoding	XMLCharEncoding	String	The character encoding to use in the XML document. The default is UTF-16 for the Unicode version of Siebel Business Applications.
Use Siebel Message Envelope	UseSiebelMessageEnvelope	String	Inserts the Siebel Message Envelope into the XML document. The default is True.
Ignore Character Set Conversion Errors	IgnoreCharSetConv Errors	String	If some characters cannot be represented in the destination character set (like the local code page), then the errors can be ignored. The errors are not ignored by default. For both situations, a warning error entry is created.
Tags on Separate Lines	Tags on Separate Lines	String	Default is True, which means that a line feed is placed at the end of each tag. If False, then no line feed is added to the end of each tag; the XML message is generated in a single line.
XML Header Text	XMLHeaderText	String	Text to prepend to the beginning of the XML document data.
Generate Namespace Declarations	GenerateNamespaceDecl	String	Default is False. If True, then the namespace declarations will be generated.
Generate Processing Instructions	GenerateProcessingInstructions	String	Default is True. If set to False, then the Siebel processing instructions are not written.

Table 17. Integration Object Hierarchy to XML Document Method Input Arguments

Display Name	Name	Data Type	Description
Generate Schema Types	GenerateSchemaTypes	String	Default is False. If set to True, then XSD schema types will be generated if set on the integration objects user properties.
Namespace	Namespace	String	If a namespace is defined here, then it will override any namespace defined in the user properties of an integration object.

Table 18 describes the output argument for the Integration Object Hierarchy to XML Document method of the EAI XML Converter.

Table 18. Integration Object Hierarchy to XML Document Method Output Argument

Display Name	Name	Data Type	Description
XML Document	<Value>	String	The resulting XML document.

XML Document to Integration Object Hierarchy Method Arguments

Table 19 describes the input arguments for the XML Document to Integration Object Hierarchy method of the EAI XML Converter.

Table 19. XML Document to Integration Object Hierarchy Method Input Arguments

Display Name	Name	Data Type	Description
XML Document	<Value>	String	The input XML document.
Integration Object Name	IntObjectName	String	Name of the Integration Object to use in cases where the Siebel Message envelope might not be present.
Integration Object Lookup Rule Set	IntObjectNameLookupRuleSet	String	Rule Set for the EAI Dispatcher Service for finding out Integration Object Name in cases where the Siebel Message envelope might not be present.

Table 19. XML Document to Integration Object Hierarchy Method Input Arguments

Display Name	Name	Data Type	Description
Validate External Entity	ValidateExternalEntity	String	If set to True, then the parser will be set to validate against external metadata, such as DTDs. If set to True, then the DOCTYPE definition must be included in the incoming XML header, for example: `<!DOCTYPE SiebelMessage SYSTEM "c:\temp\ListOfMyInbound.dtd">`
External Entity Directory	ExternalEntityDirectory	String	The directory to use for finding external entities referenced in the XML document, such as DTDs.
Truncate Field Values	TruncateFieldValues	String	Default is False. If True, then truncate any fields longer than their maximum size, as specified in the Integration Component field definition.
Ignore Character Set Conversion Errors	IgnoreCharSetConvErrors	String	Default is False. If the Siebel application cannot represent a given character set, such as the local code page character set, then conversion errors are logged, including a warning log entry. When set to True, only a warning message is logged.
Contains Inline Attachments	ContainsInlineAttachments	String	This is True if the file attachment content was included in the original XML document. Otherwise it is False. From MIME (Multipurpose Internet Mail Extensions) Converter only.
Process Elements Only	ProcessElementsOnly	String	Default is False. If set to True, then processing of attributes is skipped.

Table 20 describes the output arguments for the XML Document to Integration Object Hierarchy method of the EAI XML Converter.

Table 20. XML Document to Integration Object Hierarchy Method Output Arguments

Display Name	Name	Data Type	Description
Siebel Message	SiebelMessage	Hierarchy	The Integration Object Hierarchy to be converted to XML.
XML Character Encoding	XMLCharEncoding	String	Character encoding of the XML document, detected by the converter independent of the parser.

XML Hierarchy Converter

The XML Hierarchy Converter does not use integration object metadata, but instead relies on simple rules for converting between an XML hierarchy and an XML document. The important distinction between this service and the XML Converter is a Property Set of type XMLHierarchy, which is always presented as a child of Service Method Arguments and as a parent of the XML document root element.

As shown in Figure 5, every XML element becomes a property set where the XML tag name becomes the Type. For example, the XML element `Contact` becomes a property set of the type Contact in Siebel application. In addition, every XML attribute becomes a property within the element's property set. For example, if the attribute of the XML element "`Contact`" is `City = "Toronto"`, then "City=Toronto" will be a property for Contact.

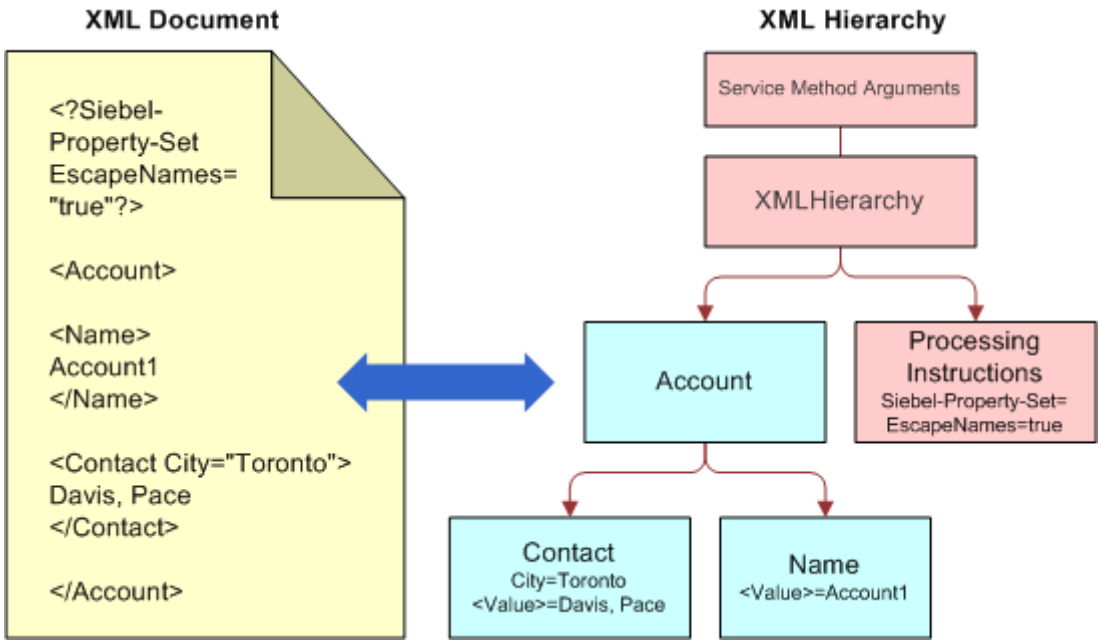

Figure 5. XML Hierarchy Representation of XML Document Structure

The convenience of having this representation is that the XML Hierarchy Converter can convert to and from this representation in the same way, independent of whether or not the XML document includes a Siebel Message or an external XML document. This representation is also handled in Siebel Workflow because it allows all the XML documents in memory to be treated as the Hierarchical Service Parameter of type XMLHierarchy.

XML Hierarchy Representation in Siebel Business Applications

- As illustrated in Figure 5 on page 52, there is a Property Set of type XMLHierarchy that always appears as a child of the Service Method Argument and the parent of the root XML element.
- Elements are represented by Property Sets. The XML tag is the type in the property set and the value assigned to that XML tag is the Value in the property set. For example, if an XML element has a value such as <Contact City="Toronto">Davis, Pace</Contact> as shown in Figure 5 on page 52, then the Value in the property set would be set to Davis, Pace and the Type in the property set would be set to contact.
- Attributes are represented as properties on the Property Set that represent the attribute's element.

- Child elements are represented as child property sets and Parent elements as Parent property sets.
- Processing instructions are represented as a child Property Set of type ProcessingInstructions, which is at the same level as the root element (the child of XML Hierarchy). In Figure 5 on page 52, the root element is Account.

The following topics are also described:

- "XML Hierarchy Converter Business Service Methods" on page 53
- "XML Document to XML Hierarchy Method Arguments" on page 54
- "XML Hierarchy to XML Document Method Arguments" on page 55

XML Hierarchy Converter Business Service Methods

There are two methods for the XML Hierarchy Converter, as shown in Table 21. The arguments for each method appear in Table 22 on page 54, Table 23 on page 55, Table 24 on page 55, and Table 25 on page 57.

Table 21. XML Hierarchy Converter Methods

Display Name	Name	Description
XML Document to XML Hierarchy	XMLDocToXMLHier	Converts an XML document into an XML Hierarchy.
XML Hierarchy to XML Document	XMLHierToXMLDoc	Converts an XML Hierarchy into an XML document.

XML Document to XML Hierarchy Method Arguments

Table 22 describes the input arguments for the XML Document to XML Hierarchy method of the XML Hierarchy Converter.

Table 22. XML Document to XML Hierarchy Method Input Arguments

Display Name	Name	Data Type	Description
XML Document	<Value>	String	The input XML Document. If XML converter business services that expect XML Document (EAI XML Converter, XML Converter, XML Hierarchy Converter) are being used, then <Value> should contain a binary buffer rather than a text string. With workflows, use the Binary data type for the process property for XML Document.
Escape Names	EscapeNames	String	Invalid characters in XML tags will be escaped, using Siebel's internal escape format. If True, then process Escape characters (this is the default). If False, then do not process Escape characters.
Validate External Entity	ValidateExternalEntity	String	If True, then the parser will be set to validate against external metadata, such as DTD schemas.
External Entity Directory	ExternalEntityDirectory	String	Location of external entity files, such as DTD files.
Ignore Character Set Conversion Errors	IgnoreCharSetConvErrors	String	Default is False. If the Siebel application cannot represent a given character set, such as the local code page character set, then conversion errors are logged, including a warning log entry. When set to True, only a warning message is logged.

Table 23 describes the output arguments for the XML Document to XML Hierarchy method of the XML Hierarchy Converter.

Table 23. XML Document to XML Hierarchy Method Output Arguments

Display Name	Name	Data Type	Description
XML Character Encoding	XMLCharEncoding	String	Character encoding of the XML document, detected by the converter, independent of the parser.
XML Hierarchy	XMLHierarchy	Hierarchy	The Output XML hierarchy.

XML Hierarchy to XML Document Method Arguments

Table 24 describes the input arguments for the XML Hierarchy to XML Document method of the XML Hierarchy Converter.

Table 24. XML Hierarchy to XML Document Method Input Arguments

Display Name	Name	Data Type	Description
Escape Names	EscapeNames	String	Invalid characters in XML tags will be escaped, using Siebel's internal escape format. ■ If True, then Escape invalid characters (this is the default). ■ If False, then delete invalid characters. (Do not use in XML tags.)
XML Character Encoding	XMLCharEncoding	String	Outputs the XML character encoding to use. If encoding is blank or not supported, then an error is produced.

Table 24. XML Hierarchy to XML Document Method Input Arguments

Display Name	Name	Data Type	Description
XML Header Text	XMLHeaderText	String	A string in a local code page character encoding to be inserted before the XML document's root element, after the <?xml...?> declaration. This allows custom processing instructions or an XML header to be inserted before the XML document data starts. For instance, if the header text is <myheader>data</myheader> and the XML document output without this parameter is <?xml version="1.0" encoding="UTF-8"?><account>..</account>, then the document with the XMLHeaderText included will be: <?xml version="1.0" encoding="UTF-8"?><myheader>some data</myheader><account>.......</account>
XML Hierarchy	XMLHierarchy	Hierarchy	The XML hierarchy.
Ignore Character Set Conversion Errors	IgnoreCharSetConvErrors	String	Default is False. If the Siebel application cannot represent a given character set, such as the local code page character set, then conversion errors are logged, including a warning log entry. When set to True, only a warning message is logged.
Tags on Separate Lines	Tags on Separate Lines	String	Default is True, which means that a line feed is placed at the end of each tag. If False, then no line feed is added to the end of each tag; the XML message is generated in a single line.
Generate Processing Instructions	GenerateProcessingInstructions	String	Default is True. If set to False, then the Siebel processing instructions are not written.

Table 25 describes the output argument for the XML Hierarchy to XML Document method of the XML Hierarchy Converter.

Table 25. XML Hierarchy to XML Document Method Output Argument

Display Name	Name	Data Type	Description
XML Document	<Value>	String	The output XML Document.

EAI Integration Object to XML Hierarchy Converter

The EAI Integration Object to XML Hierarchy Converter can be used if additional types of XML processing are needed, such as adding new elements, attributes, or envelopes to in-memory integration object property sets. XML Hierarchy property sets can be manipulated using eScript and Siebel VB.

The following topics are also described here:

- "EAI Integration Object to XML Hierarchy Converter Business Service Methods" on page 57
- "Integration Object Hierarchy to XML Hierarchy Method Arguments" on page 58
- "XML Hierarchy to Integration Object Hierarchy Method Arguments" on page 59

EAI Integration Object to XML Hierarchy Converter Business Service Methods

There are two methods for the EAI Integration Object to XML Hierarchy Converter, as shown in Table 26. The arguments for each method appear in Table 27 on page 58, Table 28 on page 58, Table 29 on page 59, and Table 30 on page 59.

NOTE: You can use the XML Hierarchy property sets to manipulate in memory XML hierarchies, such as to add new elements, attributes, or envelopes. An XML Hierarchy property set can be converted to and from an Integration Object property set using EAI Integration Object to XML Hierarchy Converter. An XML Hierarchy property set can be converted to and from an XML document using the XML Hierarchy Converter.

Table 26. EAI Integration Object to XML Hierarchy Converter Methods

Display Name	Name	Description
Integration Object Hierarchy to XML Hierarchy	IntObjHierToXMLHier	Converts an integration object hierarchy to an XML hierarchy.
XML Hierarchy to Integration Object Hierarchy	XMLHierToIntObjHier	Converts an XML hierarchy to an integration object.

Integration Object Hierarchy to XML Hierarchy Method Arguments

Table 27 describes the input arguments for the Integration Object Hierarchy to XML Hierarchy method of the EAI Integration Object to XML Hierarchy Converter.

Table 27. Integration Object Hierarchy to XML Hierarchy Input Arguments

Display Name	Name	Data Type	Description
Namespace	Namespace	String	If a namespace is defined here, then it will override any namespace defined in the user properties of an integration object.
Integration Object Hierarchy	SiebelMessage	Hierarchy	The integration object hierarchy to be converted.
Use Siebel Message Envelope	UseSiebelMessageEnvelope	String	Default is True. If set to True, then the Siebel Message Envelope is used in the XML Hierarchy, otherwise the Siebel Message Envelope is not included.
Generate Namespace Declarations	GenerateNamespaceDecl	String	Default is False. If set to True, then the namespace declaration will be generated.
Generate Schema Types	GenerateSchemaTypes	String	Default is False. If set to True, then XSD schema types will be generated if set on the integration objects user properties.

Table 28 describes the output argument for the Integration Object Hierarchy to XML Hierarchy method of the EAI Integration Object to XML Hierarchy Converter.

Table 28. Integration Object Hierarchy to XML Hierarchy Output Argument

Display Name	Name	Data Type	Description
XML Hierarchy	XMLHierarchy	Hierarchy	The converted integration object.

XML Hierarchy to Integration Object Hierarchy Method Arguments

Table 29 describes the input arguments for the XML Hierarchy to Integration Object Hierarchy method of the EAI Integration Object to XML Hierarchy Converter.

Table 29. XML Hierarchy to Integration Object Hierarchy Input Argument

Display Name	Name	Data Type	Description
Contains Inline Attachments	ContainsInlineAttachments	String	Default is True. DTYPE_ATTACHMENT fields are assumed to include actual attachment content. If False, then the field is treated as a reference to an external attachment.
Integration Object Name	IntObjectName	String	Integration Object Name can be specified if the Siebel Message envelope is not present in the XML hierarchy. The service generates the envelope automatically if this parameter is present.
Strip Name Space	StripNamespace	String	Removes the namespace from XML tags.
Truncate Field Values	TruncateFieldValues	String	Default is True. If True, then truncate any fields longer than their maximum size. If False, then report fields that are too long as errors.
XML Hierarchy	XMLHierarchy	Hierarchy	The hierarchy to be converted.
Process Elements Only	ProcessElementsOnly	String	Default is False. If set to True, then processing of attributes is skipped.

Table 30 describes the output argument for the XML Hierarchy to Integration Object Hierarchy method of the EAI Integration Object to XML Hierarchy Converter.

Table 30. XML Hierarchy to Integration Object Hierarchy Output Argument

Display Name	Name	Data Type	Description
Integration Object Hierarchy	SiebelMessage	Hierarchy	The converted integration object.

XML Converter

The XML converter uses no integration object metadata. The rules for converting between XML documents and property sets are essentially the same as the XML Hierarchy Converter. This service, however, does not create an XML hierarchy property set, but instead the XML document's root element becomes a Type top-level property set (for example, Service Method Arguments). The service is intended for importing and exporting hierarchical data (arguments, definitions, and so on) and for passing property set arguments to and from business services.

NOTE: When using this business service, you do not specify an output argument name. The Siebel application automatically maps the newly generated property set to the specified output process property.

Figure 6 shows the translation of an XML document into a property set representation within Siebel XML, and back again.

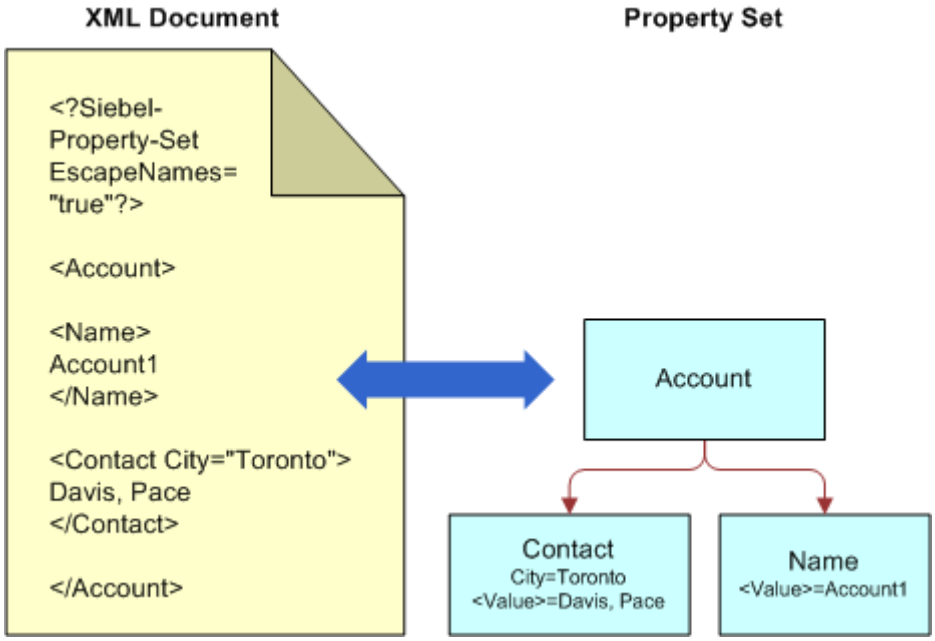

Figure 6. XML Document to Property Set Representation

The following topics are also described here:

- "XML Converter Business Service Methods" on page 60
- "Property Set To XML Method Arguments" on page 61
- "XML To Property Set Method Arguments" on page 61

XML Converter Business Service Methods

Use the XML Converter when you want to convert any property set to XML, or convert an XML document that is not a Siebel EAI Integration Object Message to a property set.

There are two methods for the XML Hierarchy Converter, as shown in Table 31. The arguments for each method appear in Table 32, Table 33, Table 34 on page 61, and Table 35 on page 62.

Table 31. XML Converter Methods

Display Name	Name	Description
Property Set to XML	PropSetToXML	Converts a property set hierarchy to XML. Returns the result in the Value field of the Output property set.
XML to Property Set	XMLToPropSet	Converts an XML document stored in the Value field of the property set to a property set hierarchy. Returns the result in the Output property set.

Property Set To XML Method Arguments

Table 32 describes the input argument for the Property Set To XML method of the XML Converter.

Table 33 describes the output argument for the Property Set To XML method of the XML Converter.

Table 32. Property Set To XML Method Input Argument

Name	Data Type	Description
Child type of the hierarchical process property containing the entire property set, service method arguments, and child property set.	Hierarchical	The entire input property set. You must manually create and name this input argument if it is required by your business needs.

Table 33. Property Set To XML Method Output Argument

Display Name	Name	Data Type	Description
XML Document	<Value>	String	The output XML document.

XML To Property Set Method Arguments

Table 34 describes the input argument for the XML To Property Set method of the XML Converter.

Table 34. XML To Property Set Method Input Argument

Display Name	Name	Data Type	Description
XML Document	<Value>	String	The input XML document

Table 35 describes the output argument for the XML To Property Set method of the XML Converter.

Table 35. XML To Property Set Method Output Argument

Name	Data Type	Description
Child type of the hierarchical process property containing the entire property set, service method arguments, and child property set.	Hierarchical	The entire output property set. You must manually create and name this output argument if it is required by your business needs.

Siebel XML Converter Business Service Comparison

Table 36 on page 62 shows the basic differences between the four XML Converter business services. The table also gives guidelines on the appropriate usage. The following terminology is used in Table 36:

- **Yes.** Supported by the converter.
- **Yes-second.** Supported when used with a second converter.

  NOTE: The EAI Integration Object to XML Hierarchy Converter always requires the XML Hierarchy Converter in the following instances.
- **No.** Not supported by the converter.

Table 36. Siebel XML Converter Comparison

Support or Requirement	EAI XML Converter	XML Hierarchy Converter	EAI Integration Object to XML Hierarchy Converter	XML Converter
Siebel Workflow	Yes	Yes	Yes-second	No
Siebel Data Mapper	Yes	Yes-second (with the EAI Integration Object Hierarchy Converter)	Yes-second	No
Siebel eScript for data transformation	Yes	Yes	Yes-second	No
Custom XML envelopes	No	Yes	Yes-second	No

Table 36. Siebel XML Converter Comparison

Support or Requirement	EAI XML Converter	XML Hierarchy Converter	EAI Integration Object to XML Hierarchy Converter	XML Converter
Dispatch Service	Yes	Yes	Yes-second	No
XML representing business service method arguments	No	No	Yes-second	Yes
Serializing property sets as XML	No	No	Yes-second	Yes
Internal representation	Siebel Message (integration object instance)	XML Hierarchy	Siebel Message (integration object instance)	Property Set
Requirement for creating an integration object definition	Yes	No	Yes	No

EAI XML Write to File Business Service

Use the EAI XML Write to File business service when you want to create an XML document from a property set hierarchy and write the resulting document to a file. This business service supports all XML converters. Table 37 describes the EAI XML Write to File business service methods.

Table 37. EAI XML Write to File Methods

Display Name	Name	Description
Write Siebel Message	WriteEAIMsg	Uses the EAI XML Converter
Write XML Hierarchy	WriteXMLHier	Uses the XML Hierarchy Converter
Write Property Set	WritePropSet	Uses the XML Converter

Write Siebel Message Method Arguments

Table 38 describes the input arguments for the Write Siebel Message method of the EAI XML Write to File business service.

Table 38. Write Siebel Message Method Input Arguments

Display Name	Name	Data Type	Description
File Name	FileName	String	The name of the file where output is to be written. This is a required field.
Siebel Message	Siebel Message	Hierarchy	The Integration Object Hierarchy to be converted to XML.
XML Character Encoding	XMLCharEncoding	String	Character encoding in the XML document. If encoding is blank or not supported, then an error is produced.
Use Siebel Message Envelope	UseSiebelMessageEnvelope	String	Default is True. Insert the Siebel Message Envelope into the XML document.
Ignore Character Set Conversion Errors	IgnoreCharSetConvErrors	String	Default is False. If the Siebel application cannot represent a given character set, such as the local code page character set, then conversion errors are logged, including a warning log entry. When set to True, only a warning message is logged.
Tags on Separate Lines	Tags on Separate Lines	String	Default is True, which means that a line feed is placed at the end of each tag. If False, then no line feed is added to the end of each tag; the XML message is generated in a single line.
Generate Namespace Declarations	GenerateNamespaceDecl	String	Default is False. If set to True, then the namespace declarations will be generated.
Generate Processing Instructions	GenerateProcessingInstructions	String	Default is True. If set to False, then the Siebel processing instructions are not written.

Table 38. Write Siebel Message Method Input Arguments

Display Name	Name	Data Type	Description
Generate Schema Types	GenerateSchemaTypes	String	Default is False. If set to True, then XSD schema types will be generated if set on the integration objects user properties.
Namespace	Namespace	String	If a namespace is defined here, it will override any namespace defined in the user properties of an integration object.

Write Property Set Method Arguments

Table 39 describes the input arguments for the Write Property Set method of the EAI XML Write to File business service.

Table 39. Write Property Set Input Arguments

Display Name	Name	Data Type	Description
File Name	FileName	String	The name of the file where output is to be written. This is a required field.
Not applicable	Child type of the hierarchical process property containing the entire property set, service method arguments, and child property set.	Hierarchical	The entire input property set. You must manually create and name this input argument if it is required by your business needs.

Write XML Hierarchy Method Arguments

Table 40 describes the input arguments for the Write XML Hierarchy method of the EAI XML Write to File business service.

Table 40. Write XML Hierarchy Method Input Arguments

Display Name	Name	Data Type	Description
File Name	FileName	String	The name of the file where output is to be written. This is a required field.
XML Hierarchy	XMLHierarchy	Hierarchy	The XML Hierarchy Property Set.

Table 40. Write XML Hierarchy Method Input Arguments

Display Name	Name	Data Type	Description
Escape Names	EscapeNames	String	Invalid characters in XML tags will be escaped, using Siebel's internal escape format. If True, then Escape invalid characters (this is the default). If False, then delete Escape characters.
XML Character Encoding	XMLCharEncoding	String	Outputs XML character encoding to use. If encoding is blank or not supported, then an error is produced.
Ignore Character Set Conversion Errors	IgnoreCharSetConvErrors	String	Default is False. If the Siebel application cannot represent a given character set, such as the local code page character set, then conversion errors are logged, including a warning log entry. When set to True, only a warning message is logged.
Tags on Separate Lines	Tags on Separate Lines	String	Default is True, which means that a line feed is placed at the end of each tag. If False, then no line feed is added to the end of each tag; the XML message is generated in a single line.
Generate Processing Instructions	GenerateProcessingInstructions	String	Default is True. If set to False, then the Siebel processing instructions are not written.

EAI XML Read from File Business Service

Use the EAI XML Read from File business service when you want to create a property set hierarchy in the Siebel environment from an XML document stored as a file. This business service supports both standard and EAI XML conversion.

Table 41 describes the three EAI XML Read from File business service's methods. The arguments for each method appear in the tables that follow.

Table 41. EAI XML Read from File Business Service Methods

Display Name	Name	Description
Read Siebel Message	ReadEAIMsg	Uses the EAI XML Converter
Read Property Set	ReadPropSet	Uses the XML Converter
Read XML Hierarchy	ReadXMLHier	Uses the XML Hierarchy Converter

Read Siebel Message Method Arguments

Table 42 describes the input arguments for the Read Siebel Message method of the EAI XML Read from File business service.

Table 42. Read Siebel Message Method Input Arguments

Display Name	Name	Data Type	Description
File Name	FileName	String	The name of the file to be read. This is a required field.
Integration Object Name	IntObjectName	String	Name of the Integration Object to use in cases where the Siebel Message header is not present.
Integration Object Lookup Rule Set	IntObjectLookupRuleSet	String	Rule Set for the EAI Dispatcher Service for finding the Integration Object Name in cases where the Siebel Message header is not present.
External Entity Directory	ExternalEntityDirectory	String	Directory to use for finding external entities referenced in the XML document, such as DTDs.
Truncate Field Values	TruncateFieldValues	String	Default is True. If True, then truncate any fields longer than their maximum size. If False, report fields that are too long as errors.
Ignore Character Set Conversion Errors	IgnoreCharSetConvErrors	String	Default is False. If the Siebel application cannot represent a given character set, such as the local code page character set, then conversion errors are logged, including a warning log entry. When set to True, only a warning message is logged.
Process Elements Only	ProcessElementsOnly	String	Default is False. If set to True, then processing of attributes is skipped.

Table 43 describes the output arguments for the Read Siebel Message method of the EAI XML Read from File business service.

Table 43. Read Siebel Message Method Output Arguments

Display Name	Name	Data Type	Description
Siebel Message	SiebelMessage	Hierarchy	The Integration Object Hierarchy converted from XML.
XML Character Encoding	XMLCharEncoding	String	Outputs XML character encoding to use. If encoding is blank or not supported, then an error is produced.

Read Property Set Method Arguments

Table 44 describes the input argument for the Read Property Set method of the EAI XML Read from File business service.

Table 44. Read Property Set Method Input Argument

Display Name	Name	Data Type	Description
File Name	FileName	String	The name of the file to be read. This is a required field.

Table 45 describes the output argument for the Read Property Set method of the EAI XML Read from File business service.

Table 45. Read Property Set Method Output Argument

Name	Data Type	Description
Child type of the hierarchical process property containing the entire property set, service method arguments, and child property set.	Hierarchical	The entire output property set. You must manually create and name this output argument if it is required by your business needs.

Read XML Hierarchy Method Arguments

Table 46 describes the input arguments for the Read XML Hierarchy method of the EAI XML Read from File business service.

Table 46. Read XML Hierarchy Method Input Arguments

Display Name	Name	Data Type	Description
File Name	FileName	String	The name of the XML file to read. This is a Required field.
Escape Names	EscapeNames	String	Invalid characters in XML tags will be escaped, using Siebel's internal escape format. ■ If True, then process Escape characters (this is the default). ■ If False, then do not process Escape characters.
External Entity Directory	ExternalEntityDirectory	String	Directory for external entities such as DTD files.
Ignore Character Set Conversion Errors	IgnoreCharSetConvErrors	String	Default is False. If the Siebel application cannot represent a given character set—such as the local code page character set—then conversion errors are logged, including a warning log entry. When set to True, only a warning message is logged.

Table 47 describes the output arguments for the Read XML Hierarchy method of the EAI XML Read from File business service.

Table 47. Read XML Hierarchy Method Output Arguments

Display Name	Name	Data Type	Description
XML Character Encoding	XMLCharEncoding	String	Character encoding of the XML document, detected by the converter independent of the parser.
XML Hierarchy	XMLHierarchy	Hierarchy	The XML Hierarchy property set.

8 Scenarios for Siebel EAI XML Integration

This chapter provides two business scenarios to assist you in implementing XML technologies for your organization. It includes the following topics:

Scenario 1: Process of Inbound Integration Using Siebel XML

This topic gives an example of how to set up an inbound integration using XML. You might use the integration differently, depending on your business model.

To set up the inbound integration, perform the following tasks:

Creating the XML Schema

Use the Generate Schema wizard in Siebel Tools to create an XSD or a DTD for the incoming XML. For details on using the Siebel XSD Wizard, see Chapter 5, "XML Integration Objects and the XSD Wizard." For details on using the Siebel DTD Wizard, see Chapter 6, "XML Integration Objects and the DTD Wizard."

To create the XML schema: XSD, DTD, or XDR

1. Launch Siebel Tools and navigate to the Integration Objects list.
2. Select an integration object from the list.
3. Click the Generate Schema button at the top of the Integration Objects list.
4. Complete the steps of the wizard:
   a. Select a business service from the Business Service drop-down list.
   b. Select the EAI Siebel Message Envelope Service from the Envelope drop-down list.
   c. Browse to a file location and type a file name to generate the schema—for example, `ListOfSiebelOrder.xml`—and click Save.
5. Load the schema into the external system.

Creating the Workflow

Create a new workflow using the Workflow Process Designer. For details on Siebel Workflow, see *Siebel Business Process Framework: Workflow Guide*.

To create a new workflow

1. Start a Siebel application and navigate to the Workflow Process Designer.
2. Create a new workflow that will take the XML file, convert it to Siebel XML format (if necessary) using the Siebel EAI XML Converter business service, call the EAI Data Transformation Engine to perform the data transformation, and call the Siebel Adapter to modify the Siebel Database as needed (upsert, delete, query, and so on).

   NOTE: The Siebel application uses an instance of the integration object you created to map the incoming XML data to fields (rows and columns) within the Siebel Database.
3. Test your workflow using the Workflow Process Simulator.
4. Save your workflow.

Running the Integration

In this scenario, assume that either an external application has generated Siebel XML that requires no translation or Siebel XML is XML that conforms to the Siebel XSD or DTD.

At run time, the Siebel application:

- Calls the EAI XML Adapter.
- Calls the EAI XML Converter to convert the incoming XML to a Siebel message.
- Calls the EAI Siebel Adapter and updates the Siebel Database with the new information just received from the incoming (external) XML document.

Scenario 2: Process of Outbound Integration Using External XML and an XSD or DTD

This topic gives one example of how to set up an integration based on incoming XLML that has been defined in an XSD or a DTD. You might use this integration differently, depending on your business model.

To set up the outbound integration, perform the following tasks:

- "Creating the Integration Object" on page 73
- "Mapping the Data" on page 73
- "Running the Integration" on page 73

Creating the Integration Object

Create a new external Siebel integration object. For details on creating integration objects, see *Integration Platform Technologies: Siebel Enterprise Application Integration*.

To create the Siebel integration object

1. Start Siebel Tools and select File, then New Object.
2. Select the EAI tab.
3. Double-click the Integration Object icon.
4. Complete the Integration Object Builder initial page:
   a. Select the Siebel project from the first drop-down list.
   b. Select EAI XSD or EAI DTD Wizard as the Business Service.
   c. Navigate to the path and file of the location of the XSD, DTD, or XML file that you want to use as the basis of the DTD.
5. Save the new integration object.

Mapping the Data

Use Siebel Data Mapper to map the fields in the external integration object with an internal Siebel integration object. For details on using the Siebel Data Mapper, see *Business Processes and Rules: Siebel Enterprise Application Integration*.

To map the data

1. Start a Siebel application and navigate to the Siebel Data Mapper.
2. Create the data mapping between the external integration object and an internal Siebel integration object.
3. Save the mapping.

   The new data mapping rules are now in the Siebel Database.

Running the Integration

In this scenario, assume that the external application has generated external XML and includes an associated XSD or a DTD.

At runtime, the Siebel application:

- Calls the EAI XML Converter to convert incoming XML to a Siebel Message.
- Calls the EAI Data Mapping Engine to transform the external integration object to an internal integration object.

A Using XML Files

This appendix discusses using XML files as an input as well as inserting a file attachment into the Siebel database using XML. It includes the following topics:

Using an XML Document as Input

You can use XML documents as input in a workflow, by calling business services to convert them to Siebel Property Sets and calling business services to process the data from XML documents as required. Figure 7 illustrates a sample workflow that uses the Siebel Adapter Insert or Update method.

Figure 7. Workflow Using Siebel Adapter with Upsert Method

The following is an example of a sample XML document containing employee information that will get *upserted* by the EAI Siebel Adapter in the workflow in Figure 7. Just before the EAI Siebel Adapter step in the workflow is invoked, the variable Employee Message will contain the XML document in a hierarchical format.

```
<SiebelMessage MessageId="" IntObjectName="Sample Employee">
  <ListOfSampleEmployees>
    <Employee>
      <FirstName>Pace</FirstName>
      <MiddleName></MiddleName>
      <LastName>Davis</LastName>
      <LoginName>ADIOTATI</LoginName>
      <PersonalTitle>Mr.</PersonalTitle>
      <EMailAddr>pdavis@pcssiebel.com</EMailAddr>
```

```
        <JobTitle>Field Sales Representative</JobTitle>
        <Phone>4153296500</Phone>
        <Private>N</Private>
          <ListOfPosition>
            <Position>
              <Name3>Field Sales Representative - S America</Name3>
              <Division>North American Organization</Division>
              <Organization>North American Organization</Organization>
              <ParentPositionName>VP Sales</ParentPositionName>
              <PositionType>Sales Representative</PositionType>
                <ListOfPosition_BusinessAddress>
                  <Position_BusinessAddress>
                  <City>San Mateo</City>
                  <Country>USA</Country>
                  <FaxNumber></FaxNumber>
                  <PhoneNumber></PhoneNumber>
                  <PostalCode>94175</PostalCode>
                  <State>CA</State>
                  <StreetAddress>1855 South Grant St</StreetAddress>
                </Position_BusinessAddress>
              </ListOfPosition_BusinessAddress>
            </Position>
          </ListOfPosition>
        </Employee>
      </ListOfSampleEmployees>
    </SiebelMessage>
```

This EAI XML document shows an integration object called Sample Employee as specified by the IntObjectName attribute of the Siebel Message element.

The Sample Employee object has three integration components you can view using Siebel Tools:

- Employee—A root component

- Position—A Child of Employee
- Position Business Address—A Child of Position

An upsert to this integration object is determined by the user key on the root component. In the Sample Employee Integration object provided as part of the sample database, the user key for the Employee integration object is Login name. Therefore, if the login name is unique, then a new employee is inserted. If the login name already exists, then the Siebel application performs an update. The above XML document will create a new employee whose name is Pace Davis and assign the position Field Sales Representative - S America to this person. You could also specify a new position and have the employee be assigned to the new position. This can be extended to other methods such as Delete or Query. If you want to delete an employee, then the user key is the only element that must be specified.

Example. In the following example, the employee with login name ADD1 will be deleted.

```
<SiebelMessage MessageId="" IntObjectName="Sample Employee">
      <ListOfSampleEmployees>
         <Employee>
            <LoginName>ADD1</LoginName>
         </Employee>
      </ListOfSampleEmployees>
   </SiebelMessage>
```

Example. Query on all employees with the first name Pace and Last name starting with D.

```
<SiebelMessage MessageId="" IntObjectName="Sample Employee">
      <ListOfSampleEmployees>
         <Employee>
            <FirstName>Pace</FirstName>
            <LastName>D*</LastName>
         </Employee>
      </ListOfSampleEmployees>
   </SiebelMessage>
```

CAUTION: When defining these business components, be aware that the precise definition can negatively affect mobile clients and regional clients. There are setup options to allow all attachments to automatically download to mobile clients that have visibility to the underlying row. This could be quite problematic, especially for large files.

The preferred setup is *demand mode*, whereby mobile client users trying to open an attachment will see a message asking if they want to download the file the next time they synchronize. This is known as the deferred approach and gives users control over what files they do or do not download.

Inserting File Attachments Using XML

You might want to insert an attachment into the Siebel Database, such as an image file in JPEG format. This could be a customer's picture, a site picture, an item or part image, a text document, and so on. For integration with external systems using file attachments, see *Integration Platform Technologies: Siebel Enterprise Application Integration*.

For integration between Siebel instances, the support for attachments is built into the Siebel Adapter and the EAI XML Converter. The integration between Siebel instances can occur when generating or reading XML, which is further defined in the next topic.

- **Generating XML.** In the case of the Attachment business component being used, the Siebel Adapter will correctly perform the query. Then, the EAI XML Converter will include the attachment in XML.
- **Reading XML.** If XML was generated by the EAI XML Converter as described previously, then the EAI XML Converter will read such XML and correctly bring attachments into memory. After which, the Siebel Adapter will insert them into Oracle's Siebel database.

Removing Empty XML Tags

You can to remove empty XML tags from messages for optimization. For example, an XML representation of an integration object might have unused integration components. You can use the siebel_ws_param:RemoveEmptyTags parameter to remove empty tags when making Web service calls.

There are two ways to use the parameter:

- "Adding the RemoveEmptyTags Parameter to a Property Set in an Input XML File" on page 78
- "Adding the RemoveEmptyTags Parameter as a Process Property in a Workflow" on page 79

Adding the RemoveEmptyTags Parameter to a Property Set in an Input XML File

You add the siebel_ws_param:RemoveEmptyTags parameter to an input XML file manually as a property in the top-level property set.

To add the RemoveEmptyTags parameter to a property set manually

1. Open the XML file in a text editor.
2. Add the following text (in bold) to the top-level <PropertySet> tag, as in this example:

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

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

<PropertySet siebel_undws_undparam_clnRemoveEmptyTags="Y">
```

```
        <SiebelMessage>
          ...
          ...
        </SiebelMessage>

    </PropertySet>
```

3 Save the XML file.

Adding the RemoveEmptyTags Parameter as a Process Property in a Workflow

You can add the siebel_ws_param:RemoveEmptyTags parameter to a workflow to automate the removal of empty tags. You add the parameter as a process property of the workflow, then as an input argument to the step that reads the XML file. For information on adding workflow process properties and input arguments, see *Siebel Business Process Framework: Workflow Guide*.

To add the RemoveEmptyTags parameter to a workflow

1 In Siebel Tools, edit the workflow process to add the following process property:

Name	Data Type	In/Out
Remove Empty Tags	String	In

2 Add the following input argument to the workflow step that reads the XML file:

Input Argument	Type	Value	Property Name
siebel_ws_param:RemoveEmptyTags	Process Property	Y	Remove Empty Tags

3 Compile the SRF.

B Sample XML for Siebel EAI Effective Dating Operations

This appendix provides sample XML for Siebel Enterprise Applications Integration (Siebel EAI) effective dating operations. It includes the following topics:

About Siebel EAI Effective Dating Operations

The Siebel Enterprise Applications Integration (Siebel EAI) effective dating framework allows access to effective dating data through various Siebel EAI communication mechanisms. You can use the typical query, insert, update, synch, and so on operations to manipulate effective dating enabled data. For more information about the Siebel effective dating feature, see *Integration Platform Technologies: Siebel Enterprise Application Integration*.

Related Topics

"Sample XML for Field-Related Siebel EAI Effective Dating Operations"

"Sample XML for Link-Related Siebel EAI Effective Dating Operations"

Sample XML for Field-Related Siebel EAI Effective Dating Operations

This topic provides sample input and output XML for field-related Siebel Enterprise Applications Integration (Siebel EAI) effective dating operations. It includes the following information:

NOTE: Bold text in the following code samples indicates syntax specific to Siebel EAI effective dating functionality.

Insert Field-Related Operations

The following code shows sample input and output XML for field-related INSERT operations.

Input

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:asi="http://example.com/asi/" xmlns:hous="http://www.example.com/xml/
Household%20Interface">

<soapenv:Header/>
<soapenv:Body>

   <asi:SiebelHouseholdInsert_Input>

         <hous:ListOfHouseholdInterface>

            <hous:Household>

               <hous:Category>Gold</hous:Category>

               <hous:CurrencyCode>USD</hous:CurrencyCode>

               <hous:EDListOfHouseholdName>
                  <hous:HouseholdName EDStartDate="04/10/2012" EDEndDate="04/20/
      2012">Adam</hous:HouseholdName>
                   <hous:HouseholdName>Becham</hous:HouseholdName>
               </hous:EDListOfHouseholdName>

               <hous:HouseholdId>ASDQ-1264</hous:HouseholdId>

               <hous:Income>47751</hous:Income>

               <hous:PhoneNumber>6504234234</hous:PhoneNumber>

               <hous:Segment>White Collar</hous:Segment>

               <hous:Status>Active</hous:Status>

               <hous:Type>Single</hous:Type>

            </hous:Household>

         </hous:ListOfHouseholdInterface>

         <asi:StatusObject>?</asi:StatusObject>

   </asi:SiebelHouseholdInsert_Input>

</soapenv:Body>
</soapenv:Envelope>
```

Output

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">
```

```
<SOAP-ENV:Body>

   <ns:SiebelHouseholdInsert_Output xmlns:ns="http://example.com/asi/">

      <ListOfHouseholdInterface xmlns="http://www.example.com/xml/
      Household%20Interface">
         <Household operation="insert">
            <HouseholdId>ASDQ-1264</HouseholdId>
            <IntegrationId/>
         </Household>
      </ListOfHouseholdInterface>

   </ns:SiebelHouseholdInsert_Output>

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

QueryById Field-Related Operations

The following code shows sample input and output XML for field-related QueryById operations.

Input

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:asi="http://example.com/asi/">

<soapenv:Header/>
<soapenv:Body>

   <asi:SiebelHouseholdQueryById_Input>
      <asi:PrimaryRowId>1-EKCK</asi:PrimaryRowId>
   </asi:SiebelHouseholdQueryById_Input>

</soapenv:Body>
</soapenv:Envelope>
```

Output

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">

<SOAP-ENV:Body>

   <ns:SiebelHouseholdQueryById_Output xmlns:ns="http://example.com/asi/">

      <ListOfHouseholdInterface xmlns="http://www.example.com/xml/
      Household%20Interface">

         <Household>

            <Category>Gold</Category>
            <CurrencyCode>USD</CurrencyCode>
            <FaxNumber/>
            <HouseholdId>1-EKCK</HouseholdId>
```

```
      <HouseholdWealth/>
      <Income/>
      <IntegrationId/>
      <PhoneNumber>6504234234</PhoneNumber>
      <Revenue>1500</Revenue>
      <Segment>White Collar</Segment>
      <Status>Active</Status>
      <Type>Single</Type>

      <ListOfRelatedContact>
         <RelatedContact IsPrimaryMVG="Y">
            <ContactIntegrationId/>
            <MiddleName>B.</MiddleName>
            <PersonUId>1-D4U9</PersonUId>
            <PersonalContact>N</PersonalContact>
            <DateEnteredHousehold>11/12/2001 17:30:29</DateEnteredHousehold>
            <DateExitedHousehold/>
            <PrimaryOrganizationId>1-19T</PrimaryOrganizationId>
            <Relationship>Head</Relationship>

            <EDListOfFirstName>
                <FirstName EDEndDate="" EDStartDate="11/08/2001">John</
FirstName>
            </EDListOfFirstName>

            <EDListOfLastName>
                <LastName EDEndDate="" EDStartDate="11/08/2001">Devine</
LastName>
            </EDListOfLastName>
         </RelatedContact>
      </ListOfRelatedContact>

      <ListOfRelatedOrganization>
         <RelatedOrganization IsPrimaryMVG="Y">
            <OrganizationName>Millennium Retail Finance Services RF ENU</
OrganizationName>
            <OrganizationId>1-19T</OrganizationId>
            <OrganizationIntegrationId/>
         </RelatedOrganization>
      </ListOfRelatedOrganization>

      <ListOfRelatedSalesRep>
         <RelatedSalesRep IsPrimaryMVG="Y">
            <Position>Siebel Administrator</Position>
            <PositionDivision>Siebel Administration</PositionDivision>
            <PositionId>0-5220</PositionId>
            <PositionIntegrationId/>
            <Login>SADMIN</Login>
         </RelatedSalesRep>
      </ListOfRelatedSalesRep>

     <EDListOfHouseholdName>
      <HouseholdName EDEndDate="" EDStartDate="11/12/2001">Devine - San Mateo
        </HouseholdName>
     </EDListOfHouseholdName>
```

```
            <EDListOfHouseholdSize>
              <HouseholdSize EDEndDate="" EDStartDate="11/12/2001">1</HouseholdSize>
            </EDListOfHouseholdSize>

         </Household>

      </ListOfHouseholdInterface>

   </ns:SiebelHouseholdQueryById_Output>

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

QueryBy Example Field-Related Operations

The following code show sample input and output XML for field-related QueryByExample operations.

Input

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:asi="http://example.com/asi/" xmlns:hous="http://www.example.com/xml/
Household%20Interface">

<soapenv:Header/>
<soapenv:Body>

   <asi:SiebelHouseholdQueryByExample_Input>

      <hous:ListOfHouseholdInterface>
         <hous:Household>
            <hous:EDListOfHouseholdName>
               <HouseholdName EDEndDate="04/20/2012" EDStartDate="04/10/2012">
      </HouseholdName>
            </hous:EDListOfHouseholdName>
            <HouseholdId>ASDQ-1264</HouseholdId>
         </hous:Household>
      </hous:ListOfHouseholdInterface>

   </asi:SiebelHouseholdQueryByExample_Input>

</soapenv:Body>
</soapenv:Envelope>
```

Output

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">

   <SOAP-ENV:Body>

   <ns:SiebelHouseholdQueryByExample_Output xmlns:ns="http://example.com/asi/">

      <ListOfHouseholdInterface xmlns="http://www.example.com/xml/
      Household%20Interface">
```

```
            <Household>

               <Category>Gold</Category>
               <CurrencyCode>USD</CurrencyCode>
               <FaxNumber/>
               <HouseholdId>ASDQ-1264</HouseholdId>
               <HouseholdWealth/>
               <Income>47751</Income>
               <IntegrationId/>
               <PhoneNumber>6504234234</PhoneNumber>
               <Revenue/>
               <Segment>White Collar</Segment>
               <Status>Active</Status>
               <Type>Single</Type>

               <ListOfRelatedContact/>

                  <ListOfRelatedOrganization>
                     <RelatedOrganization IsPrimaryMVG="Y">
                        <OrganizationName>Default Organization</OrganizationName>
                        <OrganizationId>0-R9NH</OrganizationId>
                        <OrganizationIntegrationId/>
                     </RelatedOrganization>
                  </ListOfRelatedOrganization>

                 <ListOfRelatedSalesRep>
                     <RelatedSalesRep IsPrimaryMVG="Y">
                        <Position>Siebel Administrator</Position>
                        <PositionDivision>Siebel Administration</Position Division>
                           <PositionId>0-5220</PositionId>
                           <PositionIntegrationId/>
                           <Login>SADMIN</Login>
                       RelatedSalesRep>
                  </ListOfRelatedSalesRep>

                     <EDListOfHouseholdName>
                        <HouseholdName EDEndDate="04/20/2012" EDStartDate="04/10/
      2012">Adam</HouseholdName>
                     </EDListOfHouseholdName>

                     <EDListOfHouseholdSize>
                        <HouseholdSize EDEndDate="04/19/2012"" EDStartDate="04/10/
      2012">5</HouseholdSize>
                        <HouseholdSize EDEndDate="" EDStartDate="04/20/2012">7</
      HouseholdSize>
                     </EDListOfHouseholdSize>

               </Household>

            </ListOfHouseholdInterface>

         </ns:SiebelHouseholdQueryByExample_Output>

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

Delete Field-Related Operations

The following code shows sample input and output XML for field-related DELETE operations.

Input

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:asi="http://example.com/asi/" xmlns:hous="http://www.example.com/xml/
Household%20Interface">

<soapenv:Header/>
<soapenv:Body>

   <asi:SiebelHouseholdDelete_Input>

      <hous:ListOfHouseholdInterface>
         <hous:Household>
            <hous:EDListOfHouseholdName/>
            <HouseholdId>ASDQ-1264</HouseholdId>
         </hous:Household>
      </hous:ListOfHouseholdInterface>

   </asi:SiebelHouseholdDelete_Input>

</soapenv:Body>
</soapenv:Envelope>
```

Output

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">

<SOAP-ENV:Body>

   <ns:SiebelHouseholdDelete_Output xmlns:ns="http://example.com/asi/">

      <ListOfHouseholdInterface xmlns="http://www.example.com/xml/
      Household%20Interface">
         <Household operation="delete">
            <HouseholdId>ASDQ-1264</HouseholdId>
            <IntegrationId/>
         </Household>
      </ListOfHouseholdInterface>

   </ns:SiebelHouseholdDelete_Output>

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

Synchronize Field-Related Operations

The following code shows sample input and output XML for field-related SYNCH operations.

Input

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:asi="http://example.com/asi/" xmlns:acc="http://www.example.com/xml/
Account%20Interface">

<soapenv:Header/>
<soapenv:Body>

   <asi:SiebelAccountSynchronize_Input>

      <acc:ListOfAccountInterface>

         <acc:Account>
             acc:AccountId>88-30A85</acc:AccountId>
             <acc:Name>TESTASDP</acc:Name>
             <acc:ListOfRelatedContact>
                <acc:RelatedContact>
                   <acc:ContactId>88-30ARL</acc:ContactId>
                   <acc:EDListOfFirstName>
                      <acc:FirstName EDStartDate="04/01/2012" EDEndDate="">John</
      acc:FirstName>
                   </acc:EDListOfFirstName>
                   <acc:EDListOfLastName>
                     <acc:LastName EDStartDate="04/01/2012" EDEndDate=""> Steven</
      acc:LastName>
                   </acc:EDListOfLastName>
                </acc:RelatedContact>
             </acc:ListOfRelatedContact>
         </acc:Account>

         <acc:Account>
            <acc:AccountId>ASDQ_TY2</acc:AccountId>
            <acc:Name>TESTASDT</acc:Name>
            <acc:ListOfRelatedContact>
               <acc:RelatedContact>
                  <acc:ContactId>ASDQ_TC2</acc:ContactId>
                     <acc:EDListOfFirstName>
                        <acc:FirstName EDStartDate="04/25/2012" EDEndDate="">Sam</
      acc:FirstName>
                     </acc:EDListOfFirstName>
                     <acc:EDListOfLastName>
                      <acc:LastName EDStartDate="04/25/2012" EDEndDate="">Vincent</
      acc:LastName>
                     </acc:EDListOfLastName>
               </acc:RelatedContact>
             </acc:ListOfRelatedContact>
         </acc:Account>

      </acc:ListOfAccountInterface>

   </asi:SiebelAccountSynchronize_Input>

</soapenv:Body>
</soapenv:Envelope>
```

Output

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">

<SOAP-ENV:Body>

   <ns:SiebelAccountSynchronize_Output xmlns:ns="http://example.com/asi/">

      <ListOfAccountInterface xmlns="http://www.example.com/xml/
      Account%20Interface">

         <Account operation="update">
            <AccountId>88-30A85</AccountId>
            <IntegrationId/>
         </Account>

         <Account operation="insert">
            <AccountId>88-30GK4</AccountId>
            <IntegrationId/>
         </Account>

      </ListOfAccountInterface>

   </ns:SiebelAccountSynchronize_Output>

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

Upsert Field-Related Operations

The following code shows sample input and output XML for field-related UPSERT operations.

Input

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:asi="http://example.com/asi/" xmlns:acc="http://www.example.com/xml/
Account%20Interface">

<soapenv:Header/>
<soapenv:Body>

   <asi:SiebelAccountInsertOrUpdate_Input>

      <acc:ListOfAccountInterface>

         <acc:Account>

            <acc:AccountId>88-30A85</acc:AccountId>
            <acc:Name>TESTASDP</acc:Name>
            <acc:ListOfRelatedContact>
               <acc:RelatedContact>
                  <acc:ContactId>88-30ARL</acc:ContactId>
                  <acc:EDListOfFirstName>
                     <acc:FirstName EDStartDate="04/01/2012" EDEndDate="">John</
```

```
acc:FirstName>
            </acc:EDListOfFirstName>
            <acc:EDListOfLastName>
               <acc:LastName EDStartDate="04/01/2012" EDEndDate="">Steven</
acc:LastName>
            </acc:EDListOfLastName>
         </acc:RelatedContact>
      </acc:ListOfRelatedContact>
   </acc:Account>

   <acc:Account>
      <acc:AccountId>ASDQ_TY4</acc:AccountId>
      <acc:Name>TESTASDY</acc:Name>
      <acc:ListOfRelatedContact>
          <acc:RelatedContact>
             <acc:ContactId>ASDQ_TC4</acc:ContactId>
             <acc:EDListOfFirstName>
                <acc:FirstName EDStartDate="04/25/2012" EDEndDate="">Louis</
acc:FirstName>
             </acc:EDListOfFirstName>
             <acc:EDListOfLastName>
                <acc:LastName EDStartDate="04/25/2012" EDEndDate="">George</
acc:LastName>
             </acc:EDListOfLastName>
          </acc:RelatedContact>
      </acc:ListOfRelatedContact>
   </acc:Account>

   </acc:ListOfAccountInterface>

   <!--Optional:-->

   <asi:StatusObject>?</asi:StatusObject>

</asi:SiebelAccountInsertOrUpdate_Input>

</soapenv:Body>
</soapenv:Envelope>
```

Output

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">

<SOAP-ENV:Body>

   <ns:SiebelAccountInsertOrUpdate_Output xmlns:ns="http://example.com/asi/">

      <ListOfAccountInterface xmlns="http://www.example.com/xml/
      Account%20Interface">
```

```
            <Account operation="update">
               <AccountId>88-30A85</AccountId>
               <IntegrationId/>
            </Account>

            <Account operation="insert">
               <AccountId>88-30HDZ</AccountId>
               <IntegrationId/>
            </Account>

         </ListOfAccountInterface>

      </ns:SiebelAccountInsertOrUpdate_Output>

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

Related Topics

"About Siebel EAI Effective Dating Operations" on page 81

"Sample XML for Link-Related Siebel EAI Effective Dating Operations" on page 91

Sample XML for Link-Related Siebel EAI Effective Dating Operations

This topic provides sample input and output XML for link-related Siebel Enterprise Applications Integration (Siebel EAI) effective dating operations. It includes the following information:

NOTE: Bold text in the following code samples indicates syntax specific to Siebel EAI effective dating functionality.

Insert Link-Related Operations

The following code shows sample input and output XML for link-related INSERT operations.

Input

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:asi="http://example.com/asi/" xmlns:hous="http://www.example.com/xml/
Household%20Interface">
```

```
<soapenv:Header/>
<soapenv:Body>

<asi:SiebelHouseholdInsert_Input>

   <asi:StatusObject/>

      <hous:ListOfHouseholdInterface>
         <hous:Household operation="insert">
            <hous:HouseholdName>Aaron12</hous:HouseholdName>
            <hous:HouseholdId>1-COQ1</hous:HouseholdId>

               <hous:ListOfRelatedContact>
                  <hous:RelatedContact EDStartDate="12/12/2011" EDEndDate="">
                     <ContactId>0V-19GBUM</ContactId>
                  </hous:RelatedContact>
                  <hous:RelatedContact EDStartDate="12/12/2010" EDEndDate="12/11/
      2011">
                     <ContactId>0V-19GBUM</ContactId>
                  </hous:RelatedContact>
                  <hous:RelatedContact EDStartDate="12/12/2003" EDEndDate="12/12/
      2004">
                     <ContactId>0V-18PLL2</ContactId>
                  </hous:RelatedContact>
                  <hous:RelatedContact EDStartDate="12/12/2001" EDEndDate="12/12/
      2002">
                     <ContactId>0V-18PLL2</ContactId>
                  </hous:RelatedContact>
                  <hous:RelatedContact EDStartDate="12/12/2005" EDEndDate="12/12/
      2006">
                     <ContactId>0V-18PMMD</ContactId>
                  </hous:RelatedContact>
                  <hous:RelatedContact EDStartDate="12/12/2007" EDEndDate="12/12/
      2008">
                     <ContactId>1-AJ3J</ContactId>
                  </hous:RelatedContact>
               </hous:ListOfRelatedContact>

            </hous:Household>
         </hous:ListOfHouseholdInterface>

   </asi:SiebelHouseholdInsert_Input>

</soapenv:Body>
</soapenv:Envelope>
```

Output

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">

<SOAP-ENV:Body>

   <ns:SiebelHouseholdInsert_Output xmlns:ns="http://example.com/asi/">
```

```
        <ListOfHouseholdInterface xmlns="http://www.example.com/xml/
        Household%20Interface">
           <Household operation="insert">
              <HouseholdId>1-COQ1</HouseholdId>
              <IntegrationId/>
           </Household>
        </ListOfHouseholdInterface>

     </ns:SiebelHouseholdInsert_Output>

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

QueryByExample Link-Related Operations

The following code shows sample input and output XML for link-related QueryByExample operations.

Input

```
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:asi="http://example.com/asi/" xmlns:hous="http://www.example.com/xml/
  Household%20Interface">

  <soapenv:Header/>
  <soapenv:Body>

     <asi:SiebelHouseholdQueryByExample_Input>

        <hous:ListOfHouseholdInterface>
           <hous:Household operation="?">
              <hous:HouseholdId>1-COQ1</hous:HouseholdId>
              <hous:ListOfRelatedContact>
                 <hous:RelatedContact EDStartDate="1/1/2000" EDEndDate="">
                    <hous:ContactId/>
                 </hous:RelatedContact>
              </hous:ListOfRelatedContact>
           </hous:Household>
        </hous:ListOfHouseholdInterface>

     </asi:SiebelHouseholdQueryByExample_Input>

  </soapenv:Body>
  </soapenv:Envelope>
```

Output

```
  <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
  www.w3.org/2001/XMLSchema">

  <SOAP-ENV:Body>

     <ns:SiebelHouseholdQueryByExample_Output xmlns:ns="http://example.com/asi/">
```

```
<ListOfHouseholdInterface xmlns="http://www.example.com/xml/
Household%20Interface">

   <Household>

      <Category/>
      <CurrencyCode>USD</CurrencyCode>
      <FaxNumber/>
      <HouseholdName>Aaron12</HouseholdName>
      <HouseholdId>1-COQ1</HouseholdId>
      <HouseholdSize/>
      <HouseholdWealth/>
      <Income/>
      <IntegrationId/>
      <PhoneNumber/>
      <Revenue/>
      <Segment/>
      <Status>Active</Status>
      <Type/>
      <ListOfRelatedContact>

         <RelatedContact EDEndDate="12/12/2004" IsPrimaryMVG="N"
EDStartDate="12/12/2003">
            <ContactIntegrationId/>
            <FirstName>ANDREW</FirstName>
            <LastName>LAM</LastName>
            <MiddleName/>
            <PersonUId>0V-18PLL2</PersonUId>
            <PersonalContact>N</PersonalContact>
            <ContactId>0V-18PLL2</ContactId>
            <DateEnteredHousehold>05/16/2012 05:18:50</DateEnteredHousehold>
            <DateExitedHousehold/>
            <PrimaryOrganizationId>0-R9NH</PrimaryOrganizationId>
            <Relationship/>
         </RelatedContact>

         <RelatedContact EDEndDate="12/12/2001" IsPrimaryMVG="N"
EDStartDate="12/12/2002">
               <ContactIntegrationId/>
               <FirstName>ANDREW</FirstName>
               <LastName>LAM</LastName>
               <MiddleName/>
               <PersonUId>0V-18PLL2</PersonUId>
               <PersonalContact>N</PersonalContact>
               <ContactId>0V-18PLL2</ContactId>
               <DateEnteredHousehold>05/16/2012 05:18:50<
/DateEnteredHousehold>
               <DateExitedHousehold/>
               <PrimaryOrganizationId>0-R9NH</PrimaryOrganizationId>
               <Relationship/>
         </RelatedContact>

         <RelatedContact EDEndDate="" IsPrimaryMVG="Y" EDStartDate="12/12/
2011">
            <ContactIntegrationId/>
```

```
            <FirstName>VARUN</FirstName>
            <LastName>AJWANI</LastName>
            <MiddleName/>
            <PersonUId>0V-19GBUM</PersonUId>
            <PersonalContact>N</PersonalContact>
            <ContactId>0V-19GBUM</ContactId>
            <DateEnteredHousehold>05/16/2012 05:18:50</DateEnteredHousehold>
            <DateExitedHousehold/>
            <PrimaryOrganizationId>0-R9NH</PrimaryOrganizationId>
            <Relationship/>
         </RelatedContact>

         <RelatedContact EDEndDate="12/11/2010" IsPrimaryMVG="Y"
EDStartDate="12/12/2010">
            <ContactIntegrationId/>
            <FirstName>VARUN</FirstName>
            <LastName>AJWANI</LastName>
            <MiddleName/>
            <PersonUId>0V-19GBUM</PersonUId>
            <PersonalContact>N</PersonalContact>
            <ContactId>0V-19GBUM</ContactId>
            <DateEnteredHousehold>05/16/2012 05:18:50</DateEnteredHousehold>
            <DateExitedHousehold/>
            <PrimaryOrganizationId>0-R9NH</PrimaryOrganizationId>
            <Relationship/>
         </RelatedContact>

         <RelatedContact EDEndDate="12/12/2006" IsPrimaryMVG="N"
EDStartDate="12/12/2005">
            <ContactIntegrationId/>
            <FirstName>SARVI</FirstName>
            <LastName>ANANDAN</LastName>
            <MiddleName/>
            <PersonUId>0V-18PMMD</PersonUId>
            <PersonalContact>N</PersonalContact>
            <ContactId>0V-18PMMD</ContactId>
            <DateEnteredHousehold>05/16/2012 05:18:50</DateEnteredHousehold>
            <DateExitedHousehold/>
            <PrimaryOrganizationId>0-R9NH</PrimaryOrganizationId>
            <Relationship/>
         </RelatedContact>

         <RelatedContact EDEndDate="12/12/2008" IsPrimaryMVG="N"
EDStartDate="12/12/2007">
            <ContactIntegrationId/>
            <FirstName>Felix</FirstName>
            <LastName>Aaron</LastName>
            <MiddleName>Q</MiddleName>
            <PersonUId>1-AJ3J</PersonUId>
            <PersonalContact>N</PersonalContact>
            <ContactId>1-AJ3J</ContactId>
            <DateEnteredHousehold>05/16/2012 05:18:50</DateEnteredHousehold>
            <DateExitedHousehold/>
```

```
                  <PrimaryOrganizationId>88-14P0K</PrimaryOrganizationId>
                  <Relationship/>
               </RelatedContact>

            </ListOfRelatedContact>

            <ListOfRelatedOrganization>
               <RelatedOrganization IsPrimaryMVG="Y">
                  <OrganizationName>Default Organization</OrganizationName>
                  <OrganizationId>0-R9NH</OrganizationId>
                  <OrganizationIntegrationId/>
               </RelatedOrganization>
            </ListOfRelatedOrganization>

            <ListOfRelatedSalesRep>
               <RelatedSalesRep IsPrimaryMVG="Y">
                  <Position>Siebel Administrator</Position>
                  <PositionDivision>Siebel Administration</PositionDivision>
                  <PositionId>0-5220</PositionId>
                  <PositionIntegrationId/>
                  <Login>SADMIN</Login>
               </RelatedSalesRep>
            </ListOfRelatedSalesRep>

         </Household>

      </ListOfHouseholdInterface>

   </ns:SiebelHouseholdQueryByExample_Output>

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

QueryById Link-Related Operations

The following code shows sample input and output XML for link-related QueryById operations.

Input

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:asi="http://example.com/asi/">

<soapenv:Header/>
<soapenv:Body>

   <asi:SiebelHouseholdQueryById_Input>
      <asi:PrimaryRowId>88-30D3R</asi:PrimaryRowId>
   </asi:SiebelHouseholdQueryById_Input>

</soapenv:Body>
</soapenv:Envelope>
```

Output

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">

<SOAP-ENV:Body>

   <ns:SiebelHouseholdQueryById_Output xmlns:ns="http://example.com/asi/">

      <ListOfHouseholdInterface xmlns="http://www.example.com/xml/
      Household%20Interface">

         <Household>

            <Category/>
            <CurrencyCode>USD</CurrencyCode>
            <FaxNumber/>
            <HouseholdName>Aul_HouseHold</HouseholdName>
            <HouseholdId>1-EKB3T1</HouseholdId>
            <HouseholdSize/>
            <HouseholdWealth/>
            <Income/>
            <IntegrationId/>
            <PhoneNumber/>
            <Revenue/>
            <Segment/>
            <Status>Active</Status>
            <Type/>

            <ListOfRelatedContact>
               <RelatedContact EDEndDate="" IsPrimaryMVG="Y" EDStartDate="05/16/
      2012">
                  <ContactIntegrationId/>
                  <FirstName>VARUN</FirstName>
                  <LastName>AJWANI</LastName>
                  <MiddleName/>
                  <PersonUId>0V-19GBUM</PersonUId>
                  <PersonalContact>N</PersonalContact>
                  <ContactId>0V-19GBUM</ContactId>
                  <DateEnteredHousehold>01/01/1857 00:00:00</DateEnteredHousehold>
                  <DateExitedHousehold/>
                  <PrimaryOrganizationId>0-R9NH</PrimaryOrganizationId>
                  <Relationship/>
               </RelatedContact>

               <RelatedContact EDEndDate="" IsPrimaryMVG="N" EDStartDate="05/16/
      2012">
                  <ContactIntegrationId/>
                  <FirstName>SARVI</FirstName>
                  <LastName>ANANDAN</LastName>
                  <MiddleName/>
                  <PersonUId>0V-18PMMD</PersonUId>
                  <PersonalContact>N</PersonalContact>
                  <ContactId>0V-18PMMD</ContactId>
                  <DateEnteredHousehold>01/01/1857 00:00:00</DateEnteredHousehold>
```

```
                  <DateExitedHousehold/>
                  <PrimaryOrganizationId>0-R9NH</PrimaryOrganizationId>
                  <Relationship/>
               </RelatedContact>

               <RelatedContact EDEndDate="" IsPrimaryMVG="N" EDStartDate="05/16/
      2012">
                  <ContactIntegrationId/>
                  <FirstName>Felix</FirstName>
                  <LastName>Aaron</LastName>
                  <MiddleName>Q</MiddleName>
                  <PersonUId>1-AJ3J</PersonUId>
                  <PersonalContact>N</PersonalContact>
                  <ContactId>1-AJ3J</ContactId>
                  <DateEnteredHousehold>01/01/1857 00:00:00</DateEnteredHousehold>
                  <DateExitedHousehold/>
                  <PrimaryOrganizationId>88-14P0K</PrimaryOrganizationId>
                  <Relationship/>
               </RelatedContact>

            </ListOfRelatedContact>

            <ListOfRelatedOrganization>
               <RelatedOrganization IsPrimaryMVG="Y">
                  <OrganizationName>Default Organization</OrganizationName>
                  <OrganizationId>0-R9NH</OrganizationId>
                  <OrganizationIntegrationId/>
               </RelatedOrganization>
            </ListOfRelatedOrganization>

            <ListOfRelatedSalesRep>
               <RelatedSalesRep IsPrimaryMVG="Y">
                  <Position>Siebel Administrator</Position>
                  <PositionDivision>Siebel Administration</PositionDivision>
                  <PositionId>0-5220</PositionId>
                  <PositionIntegrationId/>
                  <Login>SADMIN</Login>
               </RelatedSalesRep>
            </ListOfRelatedSalesRep>

         </Household>

      </ListOfHouseholdInterface>

   </ns:SiebelHouseholdQueryById_Output>

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

Update Link-Related Operations

The following code shows sample input and output XML for link-related UPDATE operations.

Input

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:asi="http://example.com/asi/" xmlns:hous="http://www.example.com/xml/
Household%20Interface">

<soapenv:Header/>
<soapenv:Body>

   <asi:SiebelHouseholdUpdate_Input>

      <asi:StatusObject/>

      <hous:ListOfHouseholdInterface>
        <hous:Household operation="update">
           <hous:EDListOfHouseholdName>
              <hous:HouseholdName>Aaron Household</hous:HouseholdName>
           </hous:EDListOfHouseholdName>
           <hous:HouseholdId>1-3W8</hous:HouseholdId>
           <hous:Type>Single</hous:Type>
           <hous:ListOfRelatedContact>
              <hous:RelatedContact EDStartDate="12/12/2011">
                 <hous:ContactId>1-D4U9</hous:ContactId>
              </hous:RelatedContact>
             <hous:RelatedContact EDStartDate="12/12/2004" EDEndDate="12/12/2005">
                 <hous:ContactId>0V-18PLP2</hous:ContactId>
              </hous:RelatedContact>
             <hous:RelatedContact EDStartDate="12/12/2002" EDEndDate="12/12/2004">
               <hous:ContactId>0V-19GBUM</hous:ContactId>
            </hous:RelatedContact>
           </hous:ListOfRelatedContact>
        </hous:Household>
      </hous:ListOfHouseholdInterface>

   </asi:SiebelHouseholdUpdate_Input>

</soapenv:Body>
</soapenv:Envelope>
```

Output

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">

<SOAP-ENV:Body>

   <ns:SiebelHouseholdUpdate_Output xmlns:ns="http://example.com/asi/">

      <ListOfHouseholdInterface xmlns="http://www.example.com/xml/
      Household%20Interface">
         <Household operation="update">
            <HouseholdId>1-3W8</HouseholdId>
            <IntegrationId/>
            </Household>
         </ListOfHouseholdInterface>
```

```
      </ns:SiebelHouseholdUpdate_Output>

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

Upsert Link-Related Operations

The following code shows sample input and output XML for link-related UPSERT operations.

Input

```
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:asi="http://example.com/asi/" xmlns:hous="http://www.example.com/xml/
   Household%20Interface">

   <soapenv:Header/>

   <soapenv:Body>

         <asi:SiebelHouseholdInsertOrUpdate_Input>

            <asi:StatusObject/>

            <hous:ListOfHouseholdInterface>
               <hous:Household>
                  <hous:EDListOfHouseholdName>
                     <hous:HouseholdName>A2</hous:HouseholdName>
                  </hous:EDListOfHouseholdName>
                  <hous:HouseholdId>2</hous:HouseholdId>
                  <hous:ListOfRelatedContact>
                     <hous:RelatedContact EDStartDate="12/12/2011">
                        <hous:ContactId>1-D4U9</hous:ContactId>
                     </hous:RelatedContact>
                     <hous:RelatedContact EDStartDate="12/12/2004" EDEndDate="12/12/
         2005">
                        <hous:ContactId>0V-18PLP2</hous:ContactId>
                     </hous:RelatedContact>
                     <hous:RelatedContact EDStartDate="12/12/2002" EDEndDate="12/12/
         2004">
                        <hous:ContactId>0V-19GBUM</hous:ContactId>
                     </hous:RelatedContact>
                  </hous:ListOfRelatedContact>

               </hous:Household>

            </hous:ListOfHouseholdInterface>

      </asi:SiebelHouseholdInsertOrUpdate_Input>

   </soapenv:Body>
   </soapenv:Envelope>
```

Output

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">

<SOAP-ENV:Body>

   <ns:SiebelHouseholdInsertOrUpdate_Output xmlns:ns="http://example.com/asi/">

      <ListOfHouseholdInterface xmlns="http://www.example.com/xml/
      Household%20Interface">
         <Household operation="insert">
            <HouseholdId>2</HouseholdId>
            <IntegrationId/>
         </Household>
      </ListOfHouseholdInterface>

   </ns:SiebelHouseholdInsertOrUpdate_Output>

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

Synchronize Link-Related Operations

The following code shows sample input and output XML for link-related SYNCH operations.

Input

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:asi="http://example.com/asi/" xmlns:hous="http://www.example.com/xml/
Household%20Interface">

<soapenv:Header/>
<soapenv:Body>

   <asi:SiebelHouseholdSynchronize_Input>

      <asi:StatusObject/>

      <hous:ListOfHouseholdInterface>
          <hous:Household>
              <hous:EDListOfHouseholdName>
                 <hous:HouseholdName>123</hous:HouseholdName>
              </hous:EDListOfHouseholdName>
              <hous:HouseholdId>296-5062875</hous:HouseholdId>

              <hous:ListOfRelatedContact>
                <hous:RelatedContact EDStartDate="2/2/2010" EDEndDate="1/1/2011">
                   <hous:ContactId>04-LLSQ5</hous:ContactId>
                </hous:RelatedContact>
             </hous:ListOfRelatedContact>
          </hous:Household>
       </hous:ListOfHouseholdInterface>

   </asi:SiebelHouseholdSynchronize_Input>
```

```
</soapenv:Body>
</soapenv:Envelope>
```

Output

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema">

<SOAP-ENV:Body>

   <ns:SiebelHouseholdSynchronize_Output xmlns:ns="http://example.com.com/asi/">

      <ListOfHouseholdInterface xmlns="http://www.example.com/xml/
      Household%20Interface">
         <Household operation="update">
            <HouseholdId>296-5062875</HouseholdId>
            <IntegrationId/>
         </Household>
      </ListOfHouseholdInterface>

   </ns:SiebelHouseholdSynchronize_Output>

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

Related Topics

"About Siebel EAI Effective Dating Operations" on page 81

"Sample XML for Field-Related Siebel EAI Effective Dating Operations" on page 81

Index

X