



## **BUSINESS PROCESSES AND RULES:**

**SIEBEL eBUSINESS APPLICATION INTEGRATION  
GUIDE VOLUME IV  
MIDMARKET EDITION**

*VERSION 7.5*

12-C3RL7B

*SEPTEMBER 2002*

Siebel Systems, Inc., 2207 Bridgepointe Parkway, San Mateo, CA 94404  
Copyright © 2002 Siebel Systems, Inc.  
All rights reserved.  
Printed in the United States of America

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

The full text search capabilities of Siebel eBusiness Applications include technology used under license from Hummingbird Ltd. and are the copyright of Hummingbird Ltd. and/or its licensors.

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

Supportsoft™ is a registered trademark of Supportsoft, Inc. Other product names, designations, logos, and symbols may be trademarks or registered trademarks of their respective owners.

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

**Proprietary Information**

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

# Contents

## Introduction

|                                       |    |
|---------------------------------------|----|
| How This Guide Is Organized . . . . . | 9  |
| Additional Resources . . . . .        | 10 |
| Revision History . . . . .            | 10 |

## Chapter 1. Defining Workflows for eAI

|  |    |
|--|----|
| Sample Integration Workflows . . . . .             | 12 |
| Import Account (File) . . . . .                    | 13 |
| Export Account (File) . . . . .                    | 15 |
| Import Employee (MQSeries) . . . . .               | 18 |
| Export Employee (MQSeries) . . . . .               | 21 |
| Testing the Workflow Integration Process . . . . . | 25 |

## Chapter 2. Creating and Using Dispatch Rules

|   |    |
|---|----|
| Overview of EAI Dispatch Service . . . . .    | 28 |
| EAI Dispatch Service Rule Hierarchy . . . . . | 29 |
| EAI Dispatch Service Methods . . . . .        | 30 |
| Search Expression Grammar . . . . .           | 31 |
| Output Transformation . . . . .               | 32 |
| EAI Dispatch Service . . . . .                | 34 |
| Inbound and Outbound Requests . . . . .       | 35 |
| Implementing EAI Dispatch Service . . . . .   | 37 |
| Creating a Workflow . . . . .                 | 38 |
| Defining Rule Sets . . . . .                  | 39 |

|  |    |
|--|----|
| Defining Rules . . . . .   | 39 |
| Defining Transforms . . . . .                                      | 40 |
| Invoking a Workflow Process From an EAI Dispatch Service . . . . . | 41 |
| Testing Your EAI Dispatch Service Using Argument Tracing . . . . . | 42 |
| Differences Between EAI Dispatch Service and Workflow . . . . .    | 43 |
| ProcessAggregateRequest Method . . . . .                           | 44 |
| EAI Dispatch Service Scenarios . . . . .                           | 46 |
| Outbound Scenario . . . . .  | 46 |
| Inbound Scenario . . . . .   | 48 |
| Outbound Scenarios Using ProcessAggregateRequest . . . . .         | 49 |
| Examples of Search Expression Grammar . . . . .                    | 52 |
| Examples of Dispatch Output Property Set . . . . .                 | 54 |

### **Chapter 3. Data Mapping Using the Siebel Data Mapper**

|   |    |
|---|----|
| Overview . . . . .                            | 58 |
| EAI Data Mapping Engine . . . . .             | 60 |
| EAI Data Mapping Engine Methods . . . . .     | 60 |
| Using the EAI Data Mapping Engine . . . . .   | 61 |
| The Siebel Data Mapper . . . . .              | 62 |
| Integration Object Maps . . . . .             | 63 |
| Integration Component Maps . . . . .          | 64 |
| Integration Field Maps . . . . .              | 64 |
| Creating Data Maps . . . . .                  | 65 |
| Define Integration Objects . . . . .          | 65 |
| Determining Required Maps . . . . .           | 66 |
| Creating New Data Maps . . . . .              | 66 |
| Creating Integration Component Maps . . . . . | 68 |
| Creating Integration Field Maps . . . . .     | 69 |
| Validating the Data Map . . . . .             | 69 |

|  |    |
|--|----|
| Examples of Workflow Process . . . . .                       | 70 |
| Outbound Workflow Process . . . . .                          | 70 |
| Inbound Workflow Process . . . . .                           | 75 |
| Executing the Workflow . . . . .                             | 79 |
| EAI Data Mapping Engine Expressions . . . . .                | 80 |
| Addressing Fields in Components . . . . .                    | 84 |
| Data Mapping Scenario . . . . .                              | 85 |
| Mapping Between Siebel and an External Application . . . . . | 85 |

## **Chapter 4. Data Mapping Using Scripts**

|   |     |
|---|-----|
| Overview . . . . .                              | 88  |
| EAI Data Transformation . . . . .               | 90  |
| Setting Up a Data Transformation Map . . . . .  | 90  |
| DTE Business Service Method Arguments . . . . . | 93  |
| Map Functions . . . . .                         | 95  |
| EAIEExecuteMap() Method . . . . .               | 96  |
| The Data Transformation Functions . . . . .     | 97  |
| Siebel Message Objects and Methods . . . . .    | 98  |
| Integration Message Objects . . . . .           | 98  |
| CSSEAIIntMsgIn . . . . .                        | 99  |
| CSSEAIIntMsgOut . . . . .                       | 102 |
| Integration Object Objects . . . . .            | 104 |
| CSSEAIIntObjIn . . . . .                        | 104 |
| CSSEAIIntObjOut . . . . .                       | 106 |
| Primary Integration Component Objects . . . . . | 107 |
| CSSEAIPrimaryIntCompIn . . . . .                | 107 |
| CSSEAIPrimaryIntCompOut . . . . .               | 110 |
| Integration Component Objects . . . . .         | 113 |
| CSSEAIIntCompIn . . . . .                       | 114 |
| CSSEAIIntCompOut . . . . .                      | 117 |

|  |     |
|--|-----|
| MIME Message Objects and Methods . . . . .                     | 120 |
| CSSEAIMimeMsgIn . . . . .                                      | 120 |
| CSSEAIMimeMsgOut . . . . .                                     | 124 |
| Attachments and Content Identifiers in MIME Messages . . . . . | 127 |
| XML Property Set Functions . . . . .                           | 129 |
| Top-Level Property Set Functions . . . . .                     | 129 |
| XML Element Accessors . . . . .                                | 131 |
| Examples . . . . .   | 137 |
| EAI Value Maps . . . . .                                       | 138 |
| EAIGetValueMap Function . . . . .                              | 139 |
| CSSEAIValueMap Translate Method . . . . .                      | 140 |
| EAIGetValueMap unmappedKeyHandler Argument . . . . .           | 141 |
| EAIGetValueMap() Method . . . . .                              | 143 |
| Exception Handling Considerations . . . . .                    | 144 |
| Error Codes and Error Symbols . . . . .                        | 145 |
| Data Transformation Error Processing . . . . .                 | 145 |
| Exception Handling Functions . . . . .                         | 146 |
| Sample Siebel eScript . . . . .                                | 148 |

## **Chapter 5. Troubleshooting Techniques**

|                                     |     |
|-------------------------------------|-----|
| Service Arguments Tracing . . . . . | 152 |
|-------------------------------------|-----|

## **Index**

# Introduction

This guide explains the details of the business procedures and rules—including data transformation, data mapping, and so on—of Siebel eAI, MidMarket Edition.

**NOTE:** All Siebel MidMarket product names include the phrase *MidMarket Edition* to distinguish these products from other Siebel eBusiness Applications. However, in the interest of brevity, after the first mention of a MidMarket product in this document, the product name will be given in abbreviated form. For example, after Siebel Call Center, MidMarket Edition, has been mentioned once, it will be referred to simply as Siebel Call Center. Such reference to a product using an abbreviated form should be understood as a specific reference to the associated Siebel MidMarket Edition product, and not any other Siebel Systems offering. When contacting Siebel Systems for technical support, sales, or other issues, note the full name of the product to make sure it will be properly identified and handled.

Although job titles and duties at your company may differ from those listed in the following table, the audience for this guide consists primarily of employees in these categories:

|  |   |
|--|---|
| <b>Business Analysts</b>                 | Persons responsible for analyzing application integration challenges and planning integration solutions at an enterprise.   |
| <b>Database Administrators</b>           | Persons who administer the database system, including data loading, system monitoring, backup and recovery, space allocation and sizing, and user account management. |
| <b>Siebel Application Administrators</b> | Persons responsible for planning, setting up, and maintaining Siebel applications.  |
| <b>Siebel Application Developers</b>     | Persons who plan, implement, and configure Siebel applications, possibly adding new functionality.  |

- Siebel Integration Developers** Persons responsible for analyzing a business situation or using the analysis of a business analyst to build the integration solution at an enterprise for Siebel applications.
- Siebel System Administrators** Persons responsible for the whole system, including installing, maintaining, and upgrading Siebel applications.
- System Integrators** Persons responsible for analyzing a business situation or using an analysis of a business analyst to build the integration solution at an enterprise for specific applications, and or to develop custom solutions.

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



## How This Guide Is Organized

This book is organized in a way that presents information on discrete components of the eAI business rules and processes, such as using data transformation, using workflows, and so on as individual chapters. Additional information, as applicable, can be found in the appendices.

This book is Volume 4 of a six-volume set. The full set includes:

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

## Additional Resources

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

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

## Revision History

*Business Processes and Rules: Siebel eBusiness Application Integration Guide, Volume IV, MidMarket Edition, Version 7.5*

# Defining Workflows for eAI

# 1

This chapter describes workflow integration processes and how to use them to develop your integration projects. The chapter depends on several sample workflows that are included in the Sample database. For information on using Siebel Business Process Designer to create workflow processes and workflow policies, see *Siebel Business Process Designer Administration Guide, MidMarket Edition*.

## Sample Integration Workflows

Siebel eAI includes several sample workflows that illustrate how you can receive, process, and send integration messages. This chapter includes four of those samples, along with brief descriptions that are intended to help you understand the workflow elements specific to Siebel eAI. One of the methods of invoking a workflow process is through a workflow policy. To invoke a workflow process that contains steps that call eAI adapters from a workflow policy, you must create a workflow policy action that is based on the Run Integration Process program. For details on Siebel eAI and Run Time Events, see *Platform Technologies: Siebel eBusiness Application Integration Volume II, MidMarket Edition*.

The sample workflows described in this chapter include:

- Import Account (File), see [“Import Account \(File\)” on page 13](#).
- Export Account (File), see [“Export Account \(File\)” on page 15](#).
- Import Employee (MQSeries), see [“Import Employee \(MQSeries\)” on page 18](#).
- Export Employee (MQSeries), see [“Export Employee \(MQSeries\)” on page 21](#).

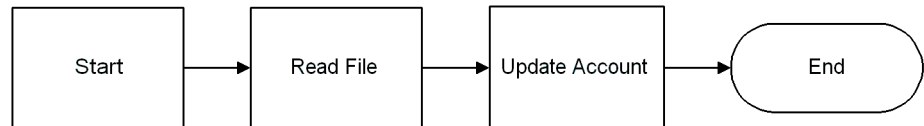
## Import Account (File)

This is a sample workflow process that reads an XML file (c:\account.xml), imports the account information into the Siebel environment, and then uses the EAI XML Read from File adapter to convert the data.

### To create a workflow to import an account

- 1 Navigate to the Workflow Process Designer.
- 2 Create a workflow with Start, End, and two business services.

The business services need to be set up for the tasks they have to accomplish.



- 3 Define the process properties.

Workflow process properties are global to the entire workflow. For example, as shown in the following table, the Import Account (File) workflow has several properties. The Account Message is defined to identify the output of the Read File step (a parsed version of the XML Account Message) as a hierarchical structure. The Error Message, Error Code, Object Id, and Siebel Operation Object Id properties are included in each workflow by default.

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

- 4 Set up the first business service, after Start, to read the information from the XML file.

This step uses the Read Siebel Message method of the EAI XML Read from File business service to convert XML from a file into an integration object hierarchy with the following Input argument.

| Input Arguments | Type    | Value          |
|-----------------|---------|----------------|
| File Names      | Literal | c:\account.xml |

Note how the path and file name are specified as a string in the Value field of the Input Arguments applet.

You also need to set up the following output property for this step.

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

- 5 Set up the second business service to update the Account in the database.

This step uses the EAI Siebel Adapter with the Insert or Update method to read the Siebel Message and insert or update the Account object in your Siebel application with the information from the XML file. This business service uses the input arguments shown below.

| Arguments      | Type             | Property Name   | Property Data Type |
|----------------|------------------|-----------------|--------------------|
| Siebel Message | Process Property | Account Message | Hierarchy          |

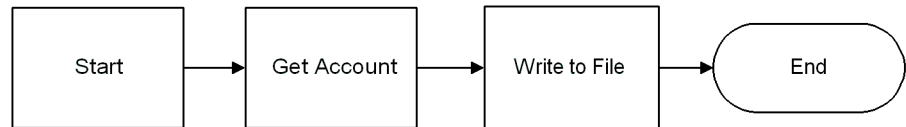
Since the Insert or Update method is specified on the EAI Siebel Adapter business service, this step checks the Siebel database to see if the Account object defined in the XML file already exists in the database. If the account exists, then it updates the account in the database with the account instance from the XML file; otherwise, it inserts the account into the database.

## Export Account (File)

This is a sample workflow process that exports an account to a file in an XML format. This workflow uses the EAI Siebel Adapter and the EAI XML Write to File adapter to convert the data from the Siebel business object to an XML document.

### **To create a workflow to export an account**

- 1** Navigate to the Workflow Process Designer.
- 2** Create a workflow with Start, End, and two business services. Set up the business services for the tasks they have to accomplish.



### 3 Define the process properties.

Workflow process properties are global to the entire workflow. For example, as shown in the following table, the Export Account (File) workflow has several properties. The Account Message is defined to identify the outbound Account as a hierarchical structure. The Error Message, Error Code, Object Id, Process Instance Id, and Siebel Operation Object Id properties are included in each workflow by default.

| Name                       | Data Type | In/Out | Default String |
|----------------------------|-----------|--------|----------------|
| Account Message            | Hierarchy | In/Out | -              |
| Error Code                 | String    | In/Out | -              |
| Error Message              | String    | In/Out | -              |
| Object Id                  | String    | In/Out | 1-6            |
| Process Instance Id        | String    | In/Out | -              |
| Siebel Operation Object Id | String    | In/Out | -              |

Note also how the Object Id process property is set to the account number 1-6, in the Default column. This string identifies an actual account in the Siebel database by its Row Id. You may set this workflow to use the active account instead of specifying a hard-coded account number. You can accomplish this by creating a button that invokes this workflow from the Account screen or you can pass the value of the Object Id into the workflow process as an input argument.



- 4 Set up the first business service, after Start, to get the account information from the database.

This step uses the EAI Siebel Adapter to query an account from your Siebel application, using the following input arguments.

| Input Arguments                | Type             | Value          | Property Name | Property Data Type |
|--------------------------------|------------------|----------------|---------------|--------------------|
| Output Integration Object Name | Literal          | Sample Account | -             | -                  |
| Object Id                      | Process Property | -              | Object Id     | String             |

Note that Output Integration Object Name of *Sample Account* is part of the query criteria. The *Sample Account* integration object describes the structure of the Account business object and was created using the Integration Object Builder. The other part of the query criteria is the Object Id, which is a process property that contains the account number 1-6 defined as a process property before.

You also need to set up the following output property for this step.

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

The output from this step is Account Message. Account Message is a process property that will contain the Siebel Message, which is the instance of Account that contains data for account number 1-6. The format is specified by the Sample Account integration object.

- 5 Set up the second business service to convert the Message to XML and write it to a file.

This step invokes the EAI XML Write to File adapter with the Write Siebel Message method using the following input arguments.

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

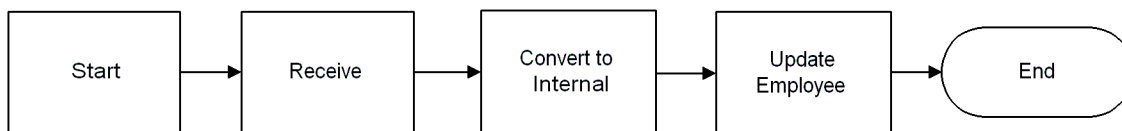
The EAI XML Write to File Adapter converts the hierarchical message to XML and writes the resulting document to the file named in the File Name argument.

## Import Employee (MQSeries)

This is a sample workflow process that receives an XML string from an IBM MQSeries queue and updates the Employee instance in the Siebel database.

### **To create a workflow to import employee using MQSeries**

- 1 Navigate to the Workflow Process Designer.
- 2 Create a workflow with Start, End, and three business services. Set up each of the business services for the task it must accomplish.



---

**NOTE:** When using the MQSeries Receiver, remember the MQ Receiver task will read the message from the queue and pass it into your Workflow Process in the < Value > field. This means your Workflow Process does not need to read the message from the MQSeries Queue. To get the XML string that has been read, you need to create a Process Property and set its default value as follows: Name = MyXMLStringProperty and Default = < Value > . You should use this Process Property as the input to the EAI XML Converter service.

---

### 3 Define the process properties.

Workflow process properties are global to the entire workflow. For example, as shown in the following table, the Import Employee (MQSeries) workflow has several properties. The Employee Message contains the object as an integration object hierarchy, when converted. The object must be in that format before it can be inserted or updated in the Siebel environment. The Employee XML property defines the MQSeries message as XML recognizable by Siebel applications. The Error Code, Error Message, Object Id, Process Instance Id, and Siebel Operation Object Id properties are included in each workflow by default.

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

- 4 Set up the first business service, after Start, to receive the Inbound Message from IBM MQSeries.

This step uses the Receive method of the EAI MQSeries Server Transport to get the inbound message from the Employee physical queue named in the Physical Queue Name argument with the following input arguments.

| Input Arguments     | Type    | Value    |
|---------------------|---------|----------|
| Physical Queue Name | Literal | Employee |
| Queue Manager Name  | Literal | Siebel   |

As shown below, the output from this step is put into the Employee XML process property with the assumption that the inbound message is already in XML format.

| Property Name | Type            | Output Argument |
|---------------|-----------------|-----------------|
| Employee XML  | Output Argument | Message Text    |

- 5 Set up the second business service to convert the inbound message.

This step uses the XML to Property Set method of the EAI XML Converter to convert the inbound message to the Siebel business object format, with the following input arguments.

| Input Arguments | Type             | Property Name | Property Data Type |
|-----------------|------------------|---------------|--------------------|
| XML Document    | Process Property | Employee XML  | Binary             |

The output from this step is passed in the Employee Message output argument as shown below.

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

- 6 Set up the third business service to update the employee record.

This step uses the Insert or Update method of the EAI Siebel Adapter with the input arguments shown below to perform the updating task.

| Input Arguments | Type             | Property Name    | Property Data Type |
|-----------------|------------------|------------------|--------------------|
| Siebel Message  | Process Property | Employee Message | Hierarchy          |

The EAI Siebel Adapter checks the Siebel database for an Employee record that matches the current instance of Employee in the Employee Message property. If an Employee record matching the current instance does not exist in the database, the EAI Siebel Adapter inserts the record into the database; otherwise, it updates the existing record with the instance.

## Export Employee (MQSeries)

This is a sample workflow process that sends an XML string for an employee to an IBM MQSeries queue.

### **To create a workflow to Export Employee using MQSeries**

- 1 Navigate to the Workflow Process Designer.
- 2 Create a workflow with Start, End, and three business services. Set up each of the business services for the task it must accomplish.



### 3 Define the process properties.

Workflow process properties are global to the entire workflow. For example, as shown in the following table, the Export Employee (MQSeries) workflow has multiple properties. The Employee Message contains the object as an integration object hierarchy, before conversion. The Employee XML property specifies the Siebel object that has been converted to XML. The Error Code, Error Message, Object Id, Process Instance Id, and Siebel Operation Object Id properties are included in each workflow by default.

| Name                       | Data Type | In/Out | Default |
|----------------------------|-----------|--------|---------|
| Employee Message           | Hierarchy | In/Out |         |
| Employee XML               | Binary    | In/Out |         |
| Error Code                 | String    | In/Out |         |
| Error Message              | String    | In/Out |         |
| Object Id                  | String    | In/Out | 1-548   |
| Process Instance Id        | String    | In/Out |         |
| Siebel Operation Object Id | String    | In/Out |         |

- 4 Set up the first business service, after Start, to query the employee record.

This step uses the Query method of the EAI Siebel Adapter with the input argument as shown below to get an instance of an employee record from the Siebel database. The Sample Employee integration object describes the structure of the Employee business object and was created using the Integration Object Builder wizard. The other part of the query criteria is the Object Id, which is a process property containing value 1-548.

| Input Arguments                | Type             | Value           | Property Name | Property Data Type |
|--------------------------------|------------------|-----------------|---------------|--------------------|
| Output Integration Object Name | Literal          | Sample Employee | -             | -                  |
| Object Id                      | Process Property | -               | Object Id     | String             |

The output from this step is passed in the Employee Message output argument as shown below.

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

- 5 Set up the second business service to convert the outbound message.

This step uses the Property Set to XML method of the EAI XML Converter to convert the outbound Siebel Message to XML and store it in the Employee XML output argument with the input argument shown below.

| Input Arguments | Type             | Property Name    | Property Data Type |
|-----------------|------------------|------------------|--------------------|
| Siebel Message  | Process Property | Employee Message | Hierarchy          |

The output from this step is passed in the Employee XML output argument as shown in the following table.

| Property Name | Type            | Value | Output Argument |
|---------------|-----------------|-------|-----------------|
| Employee XML  | Output Argument | -     | XML Document    |

- 6 Set up the third business service of the workflow to send the outbound message.

This business service invokes the EAI MQSeries Server Transport with the Send method to put the XML message onto the MQSeries queue, Employee. The message is represented by the Message Text argument, as shown in the following table.

| Input Arguments   | Type             | Value    | Property Name | Property Data Type |
|-------------------|------------------|----------|---------------|--------------------|
| Message Text      | Process Property | -        | Employee XML  | Binary             |
| PhysicalQueueName | Literal          | Employee | -             | -                  |
| QueueManagerName  | Literal          | Siebel   | -             | -                  |

The Queue Manager that handles the request is called Siebel. The XML message is put onto the Employee queue, where it remains until another application retrieves it from the queue.



## Testing the Workflow Integration Process

When you have finished defining your workflow integration process, you can use the Workflow Process Simulator to test its behavior.

---

**NOTE:** You can also enable detailed client logging and use the /s option for creating SQL spool scripts. This option provides more detailed information when running the integration workflow process in the simulator. For details, see *Siebel Remote Administration Guide, MidMarket Edition*.

---

The Workflow Process Simulator, included in the Workflow Process Manager, allows you to validate your processes before deploying them in production environments.

When you simulate an integration process that performs some external action—for example, the Export Account (File) workflow writes an XML file to a disk location—you can verify the end result by checking if the output object exists, or if a predetermined event has occurred.

### To test a process

- 1 Choose a process to simulate; for example, Export Account (File).
- 2 Choose the Process Simulator tab.
- 3 Click Start to initiate a process.

The border of the Start shape turns blue to indicate that it is the active element.

- 4 Click Next Step to initiate the next business service.

As you step through the process, the border of each active shape turns blue in turn, unless the simulator encounters an error, in which case it displays an error message alert.

- 5 Click Next Step until the simulator has processed every step and no additional steps have blue borders.

For more information about running the Process Simulator, reviewing process values, and using Workflow Process Manager and Workflow Batch Manager, see *Siebel Business Process Designer Administration Guide, MidMarket Edition*.

---

**NOTE:** Use the Process Simulator only for testing purposes. Do not use the Process Simulator to load data.

---

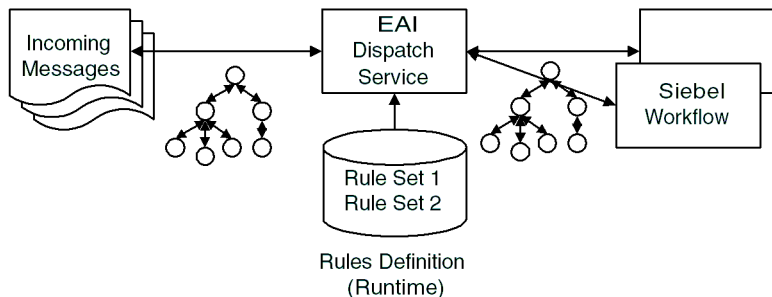
## **Creating and Using Dispatch Rules**

# **2**

This chapter gives an overview on the EAI Dispatch Service, transforming output, and implementing a new dispatch service.

## Overview of EAI Dispatch Service

The EAI Dispatch Service is a rule-based dispatching business service that invokes business services based on the properties of its input property set. EAI Dispatch Service can execute transformations on an input property set before dispatching it to the target business service. Such transformations can be useful for setting business service arguments or workflow process properties. They can also be used to do limited hierarchy manipulation such as discarding the envelope of an XML document. [Figure 1](#) illustrates the EAI Dispatch Service process.

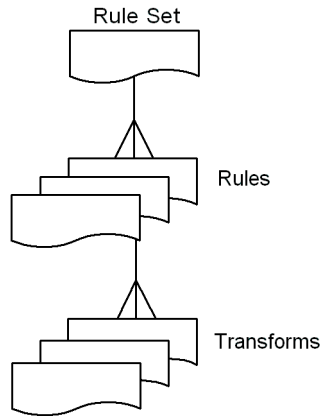


**Figure 1. EAI Dispatch Service Process**

Although the EAI Dispatch Service is a utility to invoke one business service from another business service based on specified rules, one of its primary uses is to accomplish inbound and outbound integration. The EAI Dispatch Service can be the first business service of the inbound integration to decide which business service should process an incoming document. It can also be the last step of the outbound integration to send the outgoing document to the right transport. The EAI Dispatch Service is similar to the branching in Workflow. To determine whether to use Workflow or the EAI Dispatch Service, see [“Differences Between EAI Dispatch Service and Workflow”](#) on page 43.

## EAI Dispatch Service Rule Hierarchy

The EAI Dispatch Service has a three-layer rules hierarchy as illustrated in [Figure 2](#).



**Figure 2. EAI Dispatch Service Rule Hierarchy**

### Rule Sets

Rule Sets are set of rules that you define in a particular sequence. EAI Dispatch Service parses the input document using these rules in sequence until it finds a rule that matches the input.

### Rules

Rules are individual entities in a rule set. You define rules using search expression grammar to establish how you want an input message to be routed. Each rule consists of data transformations and search expression grammar. Each rule contains zero or more rule transforms. For details on search expression grammar, see [“Search Expression Grammar” on page 31](#).

### Data Transformation

A transform specifies how the intermediate output is going to be generated before it is dispatched to the service and the method you specified in the rule. For details, see [“Output Transformation” on page 32](#).

## EAI Dispatch Service Methods

EAI Dispatch Service uses the methods described in the following table.

| Method                         | Description   |
|--------------------------------|---|
| <b>Dispatch</b>                | The Dispatch method parses the input against the rules, and then it dispatches it to the appropriate business service and business service method for further processing.   |
| <b>Lookup</b>                  | The Lookup method returns the intermediate output generation as specified by the rule output properties without dispatching it to any business service. You use this method for debugging purposes, as well as manipulating property sets within business service or workflow.  |
| <b>ProcessAggregateRequest</b> | The ProcessAggregateRequest method allows you to do multiple invocations of business services in a single request. The output for each request will be combined into a single Siebel property set or XML document. The input to this method is an XML document. For details see <a href="#">“ProcessAggregateRequest Method” on page 44</a> . |
| <b>Purge</b>                   | The Purge method clears any data that has been cached by the EAI Dispatch Service and does not take in any input arguments.   |

The EAI Dispatch Service executes the following at run time:

- Matches the input with a dispatch rule.
- Evaluates the transforms.
- Dispatches the output to a business service if the method is set to Dispatch.

## Search Expression Grammar

Search expression grammar is used by the EAI Dispatch Service to parse incoming messages and determine the course of action. Search expression grammar is based on the XPath standard. [Table 1](#) lists the definitions you use to construct a search expression.

**Table 1. Definitions for Constructing Search Expressions**

| Symbols | Description  |
|---------|--|
| /       | A forward slash indicates a new level in the hierarchy. The first slash indicates the root of the hierarchy.                   |
| @       | An at symbol indicates the attribute.  |
| *       | An asterisk indicates no specific criteria and that everything matches in the input. Asterisks cannot be used with attributes. |
| Name    | This is the literal value for which the EAI Dispatch Service searches the document.  |

**NOTE:** See [“Examples of Search Expression Grammar” on page 52](#) for additional information and examples.

## Output Transformation

Before dispatching the incoming hierarchy to the business service, EAI Dispatch Service can be used to perform some transformations to the hierarchy to make it appropriate for the target business service. A transform specifies how the intermediate output, in the memory, is going to be generated before it is dispatched to the service and the method you specified in the rule.

If you do not define any transforms, the EAI Dispatch Service will send the input directly to the business service. However, if you define transforms, the EAI Dispatch Service will create intermediate output based on the values of the transforms before sending the input to the business service you have defined in your rule.

Transforms are specified using one or more of the following targets in permissible combination.

### RootHierarchy

This target creates a new output root hierarchy based on the source expression. The source expression specifies a node in the input hierarchy. The hierarchy rooted at this node is copied as the target root hierarchy. You can use the root hierarchy for minor modifications, such as adding a property, to the input hierarchy.

Only one root hierarchy transform can be specified because this transform always creates a new hierarchy. The root hierarchy transform is always executed before any other transforms in the combination.

---

**NOTE:** For the following targets, if an output hierarchy does not exist at the time of invoking the target, an output hierarchy is first created with just an empty root node before the target is applied.

---

### ChildHierarchy

This target creates a new hierarchy as a child of the current output root hierarchy, based on the source expression. The source expression specifies a node in the input hierarchy. The hierarchy rooted at this node is copied as a new child hierarchy. You can use the child hierarchy for adding service arguments to an incoming document before dispatching to workflow or business service.



## Type

This target sets the Type field to Source Expression in the root node of output hierarchy.

## Value

This target sets the Value field to Source Expression in the root node of output hierarchy.

## Property

This target creates or overwrites a property with name Property Name and value Source Expression in the root node of output hierarchy. You can use property to add business service arguments or workflow process properties.

For certain targets, in addition to the dispatch grammar, literal values can be used for the Source Expression property to retrieve the data from the input message.

| Target         | Source Expression  | Property Name        |
|----------------|--|----------------------|
| Property       | Dispatch grammar or a literal value enclosed in quotes to search for a value | Name of the Property |
| ChildHierarchy | Grammar to search for the hierarchy  | N/A                  |
| RootHierarchy  | Grammar to search for the hierarchy  | N/A                  |
| Type           | Dispatch grammar or Literal value enclosed in quotes to search for a value   | N/A                  |
| Value          | Literal value enclosed in quotes   | N/A                  |

**NOTE:** You can combine one or more of the above transforms to achieve the desired transformation. The combination should not include more than one Root Hierarchy transform, Type transform, or Value transforms but it can include multiple Property transforms as long as the names of the properties are different.

## EAI Dispatch Service

You can use the EAI Dispatch Service to:

- Respond to a request from an external system. This can be a request to query data or a request to insert data into the Siebel database.
- Send data to an external system based on an event in Siebel applications. See [“Outbound Requests” on page 36](#).

The EAI Dispatch Service works with the hierarchy in the property set, which may be in some cases different from the hierarchy in your document. When dispatching XML documents, you should use the XML Hierarchy Converter because it generates a hierarchy matching the hierarchy in the XML document.

---

**NOTE:** For details on the XML Hierarchy Converter, see *XML Reference: Siebel eBusiness Application Integration Volume V, MidMarket Edition*.

---

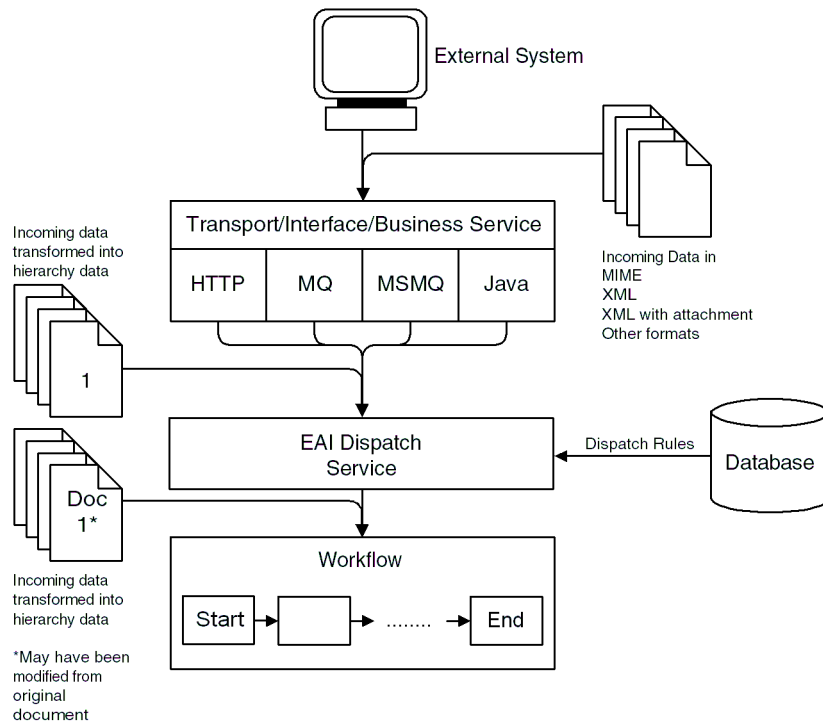
Use the business service argument tracing facility provided by the EAI Dispatch Service to understand the input property set hierarchy. This facility dumps the input and the output of the EAI Dispatch Service as XML. For details, see [“Testing Your EAI Dispatch Service Using Argument Tracing” on page 42](#).

## Inbound and Outbound Requests

The steps for creating an inbound or an outbound EAI Dispatch Service are very similar, as illustrated below.

### Inbound Requests

Figure 3 illustrates high-level architecture of an inbound EAI Dispatch Service.

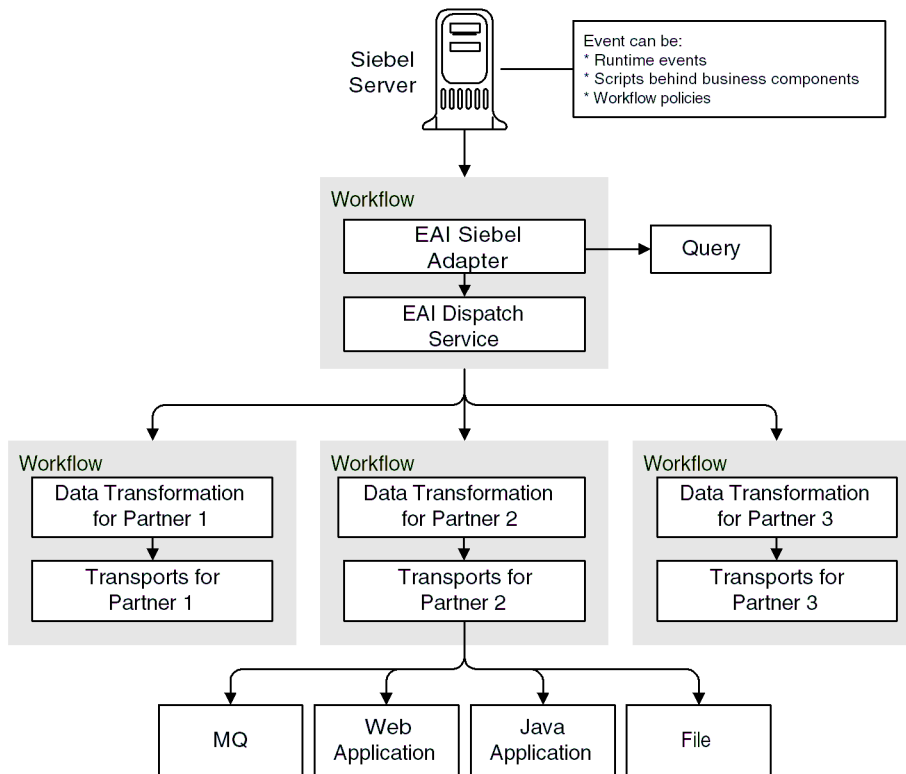


**Figure 3. Inbound EAI Dispatch Service**

### Outbound Requests

The steps for creating an outbound EAI Dispatch Service are the same as the steps for an inbound EAI Dispatch Service with some differences in the workflow.

[Figure 4](#) illustrates the high-level architecture of an outbound Dispatch Service. For details on how to create an outbound workflow, see [“Outbound Scenario” on page 46](#).



**Figure 4. Outbound EAI Dispatch Service**

## Implementing EAI Dispatch Service

The following checklist lists the steps you need to take to implement a new EAI Dispatch Service. These steps are the same whether an external system is requesting data from a Siebel application, or inserting data into a Siebel application, or when a Siebel application sends a request to an external system.

### Checklist

- 
- ☐ Create a Workflow to be called by the EAI Dispatch Service.  
For details, see [“Creating a Workflow” on page 38](#).

---

  - ☐ Define a Rule Set.  
For details, see [“Defining Rule Sets” on page 39](#).

---

  - ☐ Define Rules.  
For details, see [“Defining Rules” on page 39](#).

---

  - ☐ Define Transforms.  
For details, see [“Defining Transforms” on page 40](#).

---

  - ☐ Set up the EAI Dispatch Service to invoke the workflow.  
For details, see [“Creating a Workflow” on page 38](#).

---

  - ☐ Test your EAI Dispatch Service.  
For details, see [“Testing Your EAI Dispatch Service Using Argument Tracing” on page 42](#).
-

## Creating a Workflow

Design a workflow process to be called by EAI Dispatch Service upon receiving a request from an external system.

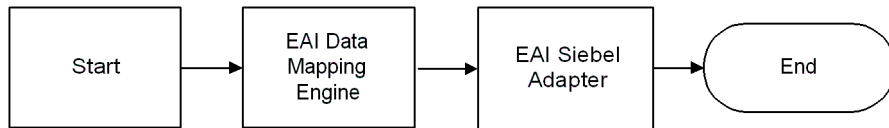
---

**NOTE:** For details on how to use Workflow Process Manager, see *Siebel Business Process Designer Administration Guide, MidMarket Edition*.

---

### **To design a workflow to receive a request from an external system**

- 1 Navigate to the Workflow Process Designer.
- 2 Set up a workflow process to include the following steps: Start, EAI Data Mapping Engine, EAI Siebel Adapter, End.



- 3 Create process properties to pass incoming data from the EAI Dispatch Service.

Because you have to pass data (as a hierarchy) from the EAI Dispatch Service to the workflow, you need to create a process property of type Hierarchy to receive this data. The name of the property should match the root tag of the hierarchy you are passing. If you use XML Hierarchy Converter with the EAI Dispatch Service, then you use the property XMLHierarchy.

Also, you may want to pass other parameters, such as what data map to use, from the EAI Dispatch Service. Create process properties of type String to receive such parameters. The name of the property should match the Property Name used in your dispatch transform.

## Defining Rule Sets

Rule sets are used by the EAI Dispatch Service to search the incoming data for specific criteria.

### To define a rule set

- 1 From the application-level menu, choose View > Site Map > Integration Administration > EAI Dispatch Service View.
- 2 Click New on the Rule Sets list applet to create a new rule set.
- 3 Give this rule set a meaningful name such as AribaAccountToSiebel.
- 4 Save the rule set.

## Defining Rules

### To define rules

- 1 Click New on the Rules list applet on the EAI Dispatch Service View.
- 2 Provide the following fields for this record:

**Sequence:** Enter a sequence number. This determines the sequence in which the application evaluates the rules.

**Search Expression:** Actual logic behind what the rule is looking for in the input. Define the Search Expression using Dispatch Rule Grammar. For details, see [“Search Expression Grammar” on page 31](#).

**Property Value (Optional):** Populate this field with the value for the property that the input is to be matched with.

**Dispatch Service:** The business service that you want to dispatch the input to. You leave this blank if you intend to use the Lookup method.

**Dispatch Method:** Pick a method for the business service you defined in the Dispatch Service field.

- 3 Save your rules.

The system validates search expression grammar. If you have not set your rules properly, you will receive an error message. See [Table 3 on page 53](#) for examples of valid search expressions.

## Defining Transforms

### To define transforms

- 1 Click New on the Transforms list applet on the EAI Dispatch Service View to create a new transform.

- 2 Provide the following fields for the new record:

**Target:** Defines how the intermediate output is going to be generated before it is dispatched to the service and the method you specified in the rule. For details, see [“Output Transformation” on page 32](#).

**Source Expression:** The source expression is used to assign a value to the target. You can either use a search expression pointing to a node in the input hierarchy or a literal value enclosed in quotes. For details, see [“Search Expression Grammar” on page 31](#).

**Property Name:** The name of the property to be set. This value is only used when the Target is set to Property. For the other Target types this field is inactive.

---

**NOTE:** See [“EAI Dispatch Service Scenarios” on page 46](#) and [“Examples of Search Expression Grammar” on page 52](#) for more details on these parameters.

---

- 3 Save your transform.

This saves and validates your transform.



## Invoking a Workflow Process From an EAI Dispatch Service

Once you created your workflow, you need to set up your EAI Dispatch Service to invoke it.

### **To invoke a workflow process with an EAI Dispatch Service**

- 1** From the application-level menu, choose View > Site Map > Integration Administration > EAI Dispatch Service View.
- 2** Select the target Rule Set.
- 3** Select the rule that invokes the workflow process.
- 4** For the selected rule set the following values:
  - **Dispatch Service.** Workflow Process Manager
  - **Dispatch Method.** Execute Process
- 5** For the selected rule insert a new record in the Transforms applet and fill in the following values:
  - **Target**—Property. You can select the Property value from a list of values.
  - **Source Expression**— < Name of the workflow process to run > . Make sure you include double quotes around the name. For example, “my workflow process.”
  - **Property Name**—Process Name. You can select the Property Name value from a list applet.

## Testing Your EAI Dispatch Service Using Argument Tracing

You should use the Business Service Simulator to test your EAI Dispatch Service before using it in your production environment. You can use argument tracing to write the input and the output of the EAI Dispatch Service as XML.

---

**NOTE:** For details on how to use the Business Service Simulator, see *Platform Technologies: Siebel eBusiness Application Integration Volume II, MidMarket Edition*.

---

### **To use the EAI dispatch service argument tracing**

- 1** Set the server parameter EnableServiceArgTracing to true.
- 2** Set the appropriate event level for EAIDispatchSvcArgTrc on your server component:
  - Event level 3.** Leads to input arguments being written out when errors occur.
  - Event level 4.** Leads to both input and output being written out.

If arguments are written out, there will be a trace log entry indicating the filename in the log directory. The filenames will have the following form:

```
<service name>_<input|output>_args_<big number>.dmp
```

For example:

```
EAIDispatchService_input_args_270613751.dmp
```

---

**NOTE:** To open the file in a XML editor, you can rename the extension to XML.

---

## Differences Between EAI Dispatch Service and Workflow

Although the EAI Dispatch Service is very similar to Workflow in initiating a task based on a condition, there are some limitations in Workflow that you can overcome using the EAI Dispatch Service. Workflow operates on business components as opposed to property sets, so Workflow can only branch based on fields in a business component. Furthermore, with Workflow you cannot route incoming documents based on property sets since the Workflow decision points cannot search inside of arbitrary property sets.

[Table 2](#) provides some guidance to help you determine the best method for your business requirements.

**Table 2. Siebel EAI Dispatch Methods and Workflow**

| Requirements  | EAI Dispatch Service | Workflow | Notes   |
|---|----------------------|----------|---|
| Need to route the incoming document based on its structure or content | ✓                    |          | The EAI Dispatch Service can route incoming documents based on property sets, whereas Workflow can only branch based on fields in a business component.           |
| Multiple dispatch targets   | ✓                    |          | The EAI Dispatch Service is a better choice because writing a workflow to include every branch can be unwieldy, but you can have many EAI Dispatch Service rules. |
| Need to change input property set before dispatching                  | ✓                    |          | The EAI Dispatch Service is the better choice since it has more powerful mapping capabilities than Workflow.  |
| Need more complex processing on the input message before dispatching  |                      | ✓        | The EAI Dispatch Service can branch based on the content of the input document, whereas Workflow can branch based on business service.                            |
| Workflow options are sufficient for your requirements                 |                      | ✓        | In this case, Workflow is the best choice.  |

## ProcessAggregateRequest Method

The ProcessAggregateRequest method allows you to perform multiple invocations of business services in a single request. The method bundles the output for each request into a single Siebel property set or XML document.

When using the ProcessAggregateRequest method with the EAI Dispatch Service business service, you need to define an input argument called AggregatedServiceRequest, with type Hierarchy for the EAI Dispatch Service to use to store the incoming data.

The following example is the input argument for this method, using XML to represent the PropertySet.

....

```
<PropertySet>
```

```
  <AggregatedServiceRequest>
```

This is the input/output method argument for the ProcessAggregatedRequest method. The EAI Dispatch Service with ProcessAggregateRequest Method looks for this XML tag within the XML document to determine where it needs to start reading the document.

```
    <BusinessServiceWrapper
```

wrapper around the business service. The name of the wrapper has no effect on the EAI Dispatch Service.

```
      BusinessServiceName=...
```

XML tag for business service

```
      BusinessServiceMethod=...>
```

XML tag for business service method

```
    <ArgumentWrapper
```

wrapper around the business service arguments. The name of the wrapper has no effect on the EAI Dispatch Service.

```
      XMLTagArgument1=...
```

XML tag for the first argument. Replace this tag with the correct XML tag for the argument your business service method is using.

```
XMLTagArgument2=...
```

XML tag for the second argument. Replace this tag with the correct XML tag for the argument your business service method is using.

```
.../>
```

```
</BusinessServiceWrapper>
```

---

**NOTE:** For examples, see [“Outbound Scenarios Using ProcessAggregateRequest” on page 49](#).

---

## EAI Dispatch Service Scenarios

The following business scenarios explain how you might accomplish commonly performed tasks using the EAI Dispatch Service.

### Outbound Scenario

For this scenario, you want to dispatch a service request as soon as it is created. The scenario assumes that:

- You are only interested in service requests logged against eAI.
- You know how to design a workflow that gets triggered as a new service request is created.

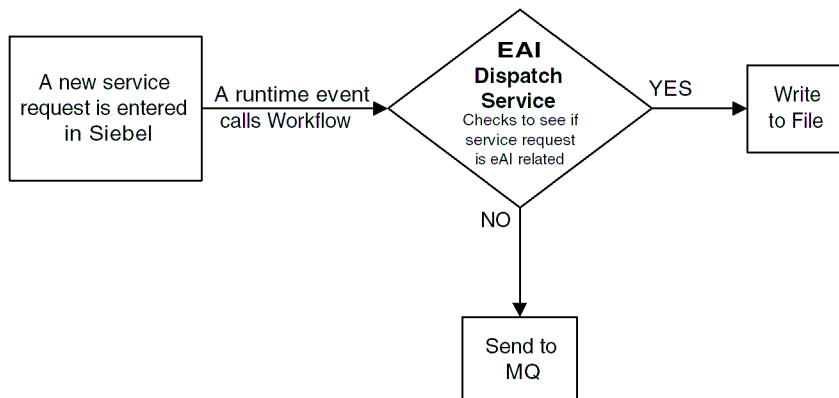
---

**NOTE:** There are number of different ways to trigger a workflow process. For details, see *Siebel Business Process Designer Administration Guide, MidMarket Edition*.

---

- You want the other non-eAI service requests to be sent to an MQSeries.

Figure 5 illustrates this scenario.



**Figure 5. Dispatching Service Request**

**To create this scenario**

- 1** Create a rule set with a search expression to check if the Service Request Area is set to eAI or not.
- 2** Create a workflow that is triggered when the criteria defined in [Step 1](#) is matched.

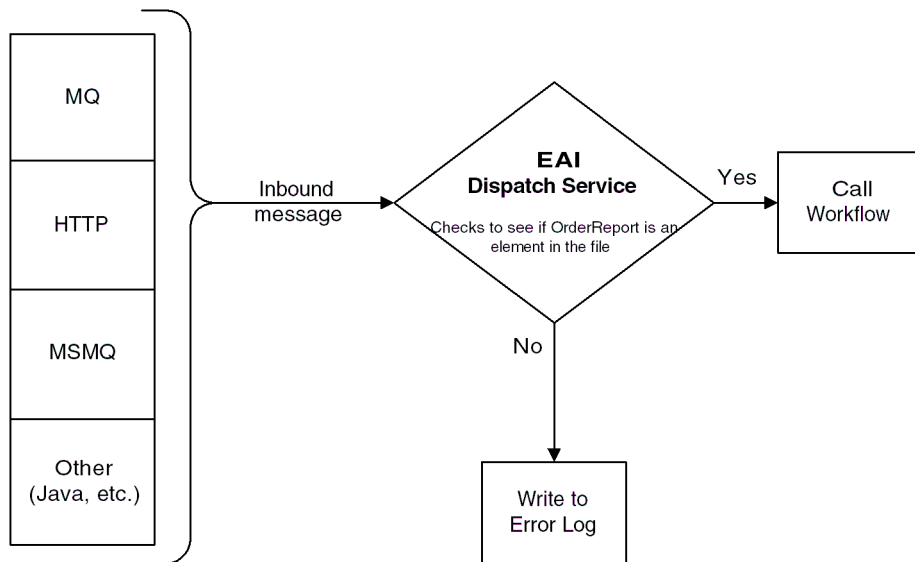
Your workflow should contain the following steps:

- Start
- EAI Dispatch Service
- End

## Inbound Scenario

For this scenario, you want to receive an XML document from an external system through MQ, HTTP, MSMQ, or other means and have the EAI Dispatch Service write to an error file if certain criteria are not met, as illustrated in [Figure 6](#). The scenario assumes that:

- You are only interested in the message if it contains an OrderReport element; otherwise, you want an error written to the error log.
- You know how to create a workflow.



**Figure 6. EAI Dispatching Service Request**



**To create this scenario**

- 1 Create a rule set with a rule that searches the message for the OrderReport element.
- 2 Create a workflow that contains the following steps:
  - Start
  - EAI Data Mapping Engine
  - EAI Siebel Adapter
  - End
- 3 Create an EAI Dispatch Service that triggers your workflow, once the criteria in [Step 1](#) are matched.

## Outbound Scenarios Using ProcessAggregateRequest

The ProcessAggregateRequest method allows you to have multiple invocation of one or more methods in one or more business services using a single request. The following examples illustrate the use of this method to query account and employee information.

### Querying the Account Integration Object

The following example shows how you can invoke multiple business services and setting arguments for each of the services. This is done using simple arguments for the services and by having the aggregate request invoke the QueryPage method of the EAI Siebel Adapter twice, with different SearchSpecs.

```
<?xml version="1.0" encoding="UTF-8"?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet>
  <AggregatedServiceRequest>
    <BusinessServiceWrapper
      BusinessServiceName="EAI Siebel Adapter"
      BusinessServiceMethod="QueryPage">
      <Argument Wrapper
        PageSize="4"
        StartRowNum="0"
        OutputIntObjectName="Sample Account"
        SearchSpec="[Account.Name] LIKE 'Aa'"/>
      </BusinessServiceWrapper>
```

```
<BusinessServiceWrapper
  BusinessServiceName="EAI Siebel Adapter"
  BusinessServiceMethod="QueryPage">
  <ArgumentWrapper
    PageSize="4"
    StartRowNum="0"
    OutputIntObjectName="Sample Account"
    SearchSpec="[Account.Name] LIKE 'Bb*'"/>
  </BusinessServiceRequest>
</AggregatedServiceRequest>
</PropertySet>
```

### Querying the Employee Integration Object

The following example shows how you can set complex type business service method arguments. The aggregate request invokes the EAI Siebel Adapter twice, and, instead of using searchspec, uses query by example by passing in a SiebelMessage.

---

**NOTE:** All simple arguments are attributes of the ArgumentWrapper element, and the complex argument is a child element.

---

```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet>
<AggregatedServiceRequest>
  <BusinessServiceWrapper
    BusinessServiceName="EAI Siebel Adapter"
    BusinessServiceMethod="Query">
    <ArgumentWrapper>
      <SiebelMessage
        MessageType="Integration Object"
        IntObjectName="Sample Employee"
        IntObjectFormat="Siebel Hierarchical">
        <ListOfSampleEmployee>
          <Employee EMailAddr="madams@siebel.com" />
        </ListOfSampleEmployee>
      </SiebelMessage>
    </ArgumentWrapper>
  </BusinessServiceWrapper>
  <BusinessServiceWrapper
    BusinessServiceName="EAI Siebel Adapter"
    BusinessServiceMethod="Query">
    <ArgumentWrapper>
      <SiebelMessage
```

```
MessageType="Integration Object"
IntObjectName="Sample Employee"
IntObjectFormat="Siebel Hierarchical">
<ListOfSampleEmployee>
  <Employee FirstName="John" LastName="Doe"/>
</ListOfSampleEmployee>
</SiebelMessage>
</ArgumentWrapper>
</BusinessServiceWrapper>
</AggregatedServiceRequest>
</PropertySet>
```

## Examples of Search Expression Grammar

In the following example, assume that the XML document is a typical document your system receives and that you want to set some rules for the EAI Dispatch Service to use to parse this document.

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

- <cXML payloadID="3223232@ariba.acme.com" timestamp="1999-03-12T18:39:09-08:00" xml:lang="en-US">

- <Header>
  - <From>
    - <Credential domain="AribaNetworkUserId">
      <Identity>admin@acme.com</Identity>
    </Credential>
    - <Credential domain="AribaNetworkUserId" type="marketplace">
      <Identity>bigadmin@marketplace.org</Identity>
    </Credential>
    - <Credential domain="BT">
      <Identity>2323</Identity>
    </Credential>
  </From>

  - <To>
    - <Credential domain="DUNS">
      <Identity>942888711</Identity>
    </Credential>
  </To>

  - <Sender>
    - <Credential domain="AribaNetworkUserId">
      <Identity>admin@acme.com</Identity>
      <SharedSecret>abracadabra</SharedSecret>
    </Credential>
    <UserAgent>Ariba.com Network V1.0</UserAgent>
  </Sender>

</Header>

  - <Request deploymentMode="test">
    -<OrderRequest>
      - <OrderRequestHeader orderID="D01234" orderDate="1999-03-12" type="new">
        - <Total>
          <Money currency="USD">12.34</Money>
        </Total>
      </OrderRequestHeader>
    </OrderRequest>
  </Request>
</cXML>
```

```

        </Total>
    - <ShipTo>
    .....
    .....

```

Table 3 provides some valid search expression examples.

**Table 3. Dispatch Rule Grammar**

| Search Expression                             | Description  |
|---|--|
| <code>/*Header</code>                         | Go to the second level and look at the type value of each property set and check whether it is of value Header.                                      |
| <code>/*/*@DeploymentMode</code>              | Go to the second level and look at the properties of each property set and check whether any of them has the name (not the value) of DeploymentMode. |
| <code>/*/*/<br/>Request@DeploymentMode</code> | Go to the third level and look at each property set for type of value Request and property of name DeploymentMode.                                   |
| <code>/cXML/*/OrderRequest</code>             | Search at the top level for type of value cXML and then upon matching, find a grandchild (not child) of type of value OrderRequest.                  |

Following are examples of invalid rules:

**Rule:** `/*/*@DeploymentMode/Request/SiebelMessage`

**Interpretation:** This is not a valid rule. A search for a property value must be specified at the very end. A correct form would be the following, which will have a different result.

```
/*/Request/*@DeploymentMode
```

**Rule:** `/*@PayLoadID@TimeStamp`

**Interpretation:** This also is not a valid rule. It is not possible to specify more than one property name. The correct form would use two different rules to represent this:

```
/*@PayLoadID
```

and

```
/*@TimeStamp
```

# Examples of Dispatch Output Property Set

This example shows different output property sets generated by EAI Dispatch Service based on the hierarchy input shown in [Figure 7](#) and certain Target and Source Expression as shown in [Table 4](#).

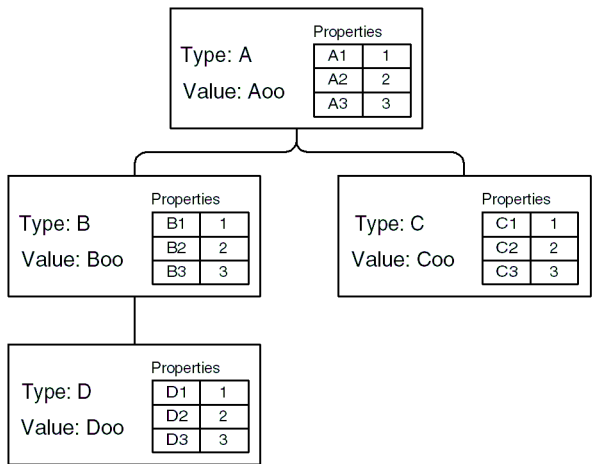
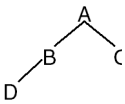




Figure 7. A Hierarchy Input

[Table 4](#) describes the intermediate output based on the value of the Target.

Table 4. Output Property Generated by EAI Dispatch Service

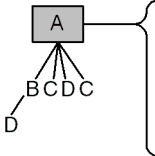
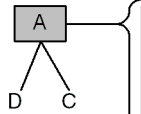
| Target        | Source Expression | Property Name | Output Property Set   |
|---------------|-------------------|---------------|---|
| RootHierarchy | /*                | N/A           |  |
| RootHierarchy | */B               | N/A           |  |
| RootHierarchy | /*/*@C1           | N/A           |  |

**Table 4. Output Property Generated by EAI Dispatch Service**

| Target         | Source Expression | Property Name | Output Property Set  |
|----------------|-------------------|---------------|--|
| ChildHierarchy | /*                | N/A           | <div><div></div><div>A</div><div>B</div><div>C</div><div>D</div></div> |
| ChildHierarchy | /**/D             | N/A           | <div><div></div><div>D</div></div>                                     |
| Type           | “abc”             | N/A           | Type : a b c   |
| Type           | /*/B              | N/A           | Type : B   |
| Type           | /**/*@B1          | N/A           | Type : 1   |
| Value          | “abc”             | N/A           | Value : a b c  |
| Property       | “foo”             | Boo           | <div>Properties<div>Boofoo</div></div>                                 |
| Property       | /**/**@D1         | Boo           | <div>Properties<div>Boo1</div></div>                                   |

You can also combine different Targets to search the input message as shown on [Table 5](#).

**Table 5. Complex Output Property Generated by EAI Dispatch Service**

| Target         | Source Expression | Property Name | Output Property  |            |  |    |      |    |      |    |   |   |      |
|----------------|-------------------|---------------|--|------------|--|----|------|----|------|----|---|---|------|
| RootHierarchy  | /*                | N/A           |  <div>Type: demo<br/>Value:</div> <table><thead><tr><th colspan="2">Properties</th></tr></thead><tbody><tr><td>A1</td><td>this</td></tr><tr><td>A2</td><td>2</td></tr><tr><td>A3</td><td>3</td></tr><tr><td>f</td><td>that</td></tr></tbody></table> | Properties |  | A1 | this | A2 | 2    | A3 | 3 | f | that |
| Properties     |                   |               |  |            |  |    |      |    |      |    |   |   |      |
| A1             | this              |               |  |            |  |    |      |    |      |    |   |   |      |
| A2             | 2                 |               |  |            |  |    |      |    |      |    |   |   |      |
| A3             | 3                 |               |  |            |  |    |      |    |      |    |   |   |      |
| f              | that              |               |  |            |  |    |      |    |      |    |   |   |      |
| ChildHierarchy | /* /* /D          | N/A           |  |            |  |    |      |    |      |    |   |   |      |
| ChildHierarchy | /* /* @C1         | N/A           |  |            |  |    |      |    |      |    |   |   |      |
| Type           | “demo”            | N/A           |  |            |  |    |      |    |      |    |   |   |      |
| Property       | “this”            | A1            |  |            |  |    |      |    |      |    |   |   |      |
| Property       | “that”            | f             |  |            |  |    |      |    |      |    |   |   |      |
| ChildHierarchy | /* /* /D          | N/A           |  <div>Type: demo<br/>Value:</div> <table><thead><tr><th colspan="2">Properties</th></tr></thead><tbody><tr><td>A1</td><td>this</td></tr><tr><td>f</td><td>that</td></tr></tbody></table>  | Properties |  | A1 | this | f  | that |    |   |   |      |
| Properties     |                   |               |  |            |  |    |      |    |      |    |   |   |      |
| A1             | this              |               |  |            |  |    |      |    |      |    |   |   |      |
| f              | that              |               |  |            |  |    |      |    |      |    |   |   |      |
| ChildHierarchy | /* /* @C1         | N/A           |  |            |  |    |      |    |      |    |   |   |      |
| Type           | “demo”            | N/A           |  |            |  |    |      |    |      |    |   |   |      |
| Property       | “this”            | A1            |  |            |  |    |      |    |      |    |   |   |      |
| Property       | “that”            | f             |  |            |  |    |      |    |      |    |   |   |      |



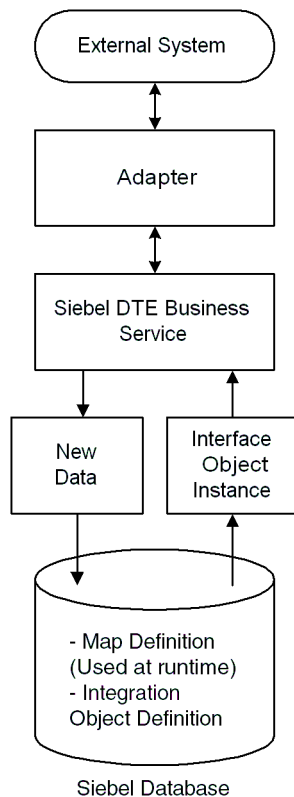
## **Data Mapping Using the Siebel Data Mapper**

# **3**

This chapter describes the process of using the Siebel Data Mapper to convert your external data to the Siebel format and your Siebel data to your external data specifications.

## Overview

The Siebel Data Mapper provides you with a declarative interface to specify maps for both inbound and outbound data transformation. The maps you set up using the Siebel Data Mapper call the EAI Data Mapping Engine to complete the data transformation. Using the Siebel Data Mapper can often reduce or even eliminate the number of scripts you need to write. [Figure 8](#) illustrates the EAI Data Mapping Engine architecture.



**Figure 8. EAI Data Mapping Engine Architecture**

For data mapping within Siebel eBusiness Applications, Siebel applications now support two data mapping solutions, the Siebel Data Mapper and Siebel eScript Data Mapping. The Siebel Data Mapper has a declarative interface and requires no programming skills. The Siebel eScript Data Mapping uses scripts programmed in eScript as data maps. Since the Siebel Data Mapper is based on a declarative interface, it does not have the flexibility that script-based data mapping has. Use Siebel Data Mapper for most of your integration needs, except for complex mapping situations requiring aggregation, joins, or programmatic flow control.

## EAI Data Mapping Engine

In order to use the EAI Data Mapping Engine, you need to enable the following component groups:

- Siebel Workflow
- Siebel eAI

## EAI Data Mapping Engine Methods

The EAI Data Mapping Engine business service has two methods: Execute and Purge.

### Execute

Use the Execute method when your integration requires data transformation. Input and output arguments for the Execute method are shown in [Table 6](#) and [Table 7](#).

**Table 6. Input Arguments for Execute Method**

| Input Argument                            | Description   |
|---|---|
| Map Name                                  | Name of your data map.  |
| Output Integration Object Name (Optional) | The target integration object in your map. If you use this argument you have to match it with the data map. |
| Siebel Message                            | The instance of your source integration object.   |
| Map Arguments (Optional)                  | Used as an argument when you call your map from a Workflow.   |

**Table 7. Output Arguments for the Execute Method**

| Property Name        | Description   |
|----------------------|---|
| Name of the property | The output integration object in Siebel Message format. |

## **Purge**

This method is only for development mode. Use the Purge method to purge the database of an existing map. Use this method when you have made a change to a map and you would like to run Execute after these changes. This method does not require any input or output arguments.

## **Using the EAI Data Mapping Engine**

The following checklist outlines the main steps required to use the EAI Data Mapping Engine.

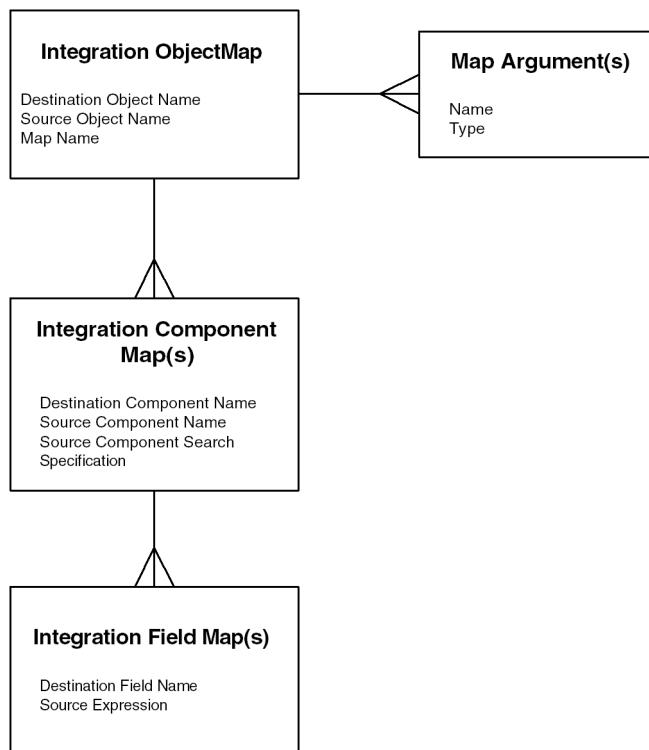
### **Checklist**

- 
- |                          |  |
|--------------------------|--|
| <input type="checkbox"/> | Create integration objects.<br>For details, see <a href="#">“Define Integration Objects” on page 65.</a> |
| <hr/>                    |  |
| <input type="checkbox"/> | Create data maps.<br>For details, see <a href="#">“Creating New Data Maps” on page 66.</a>               |
| <hr/>                    |  |
| <input type="checkbox"/> | Validate data maps.<br>For details, see <a href="#">“Validating the Data Map” on page 69.</a>            |
-

# The Siebel Data Mapper

The Siebel Data Mapper maps one integration object, source, to another integration object, target. Integration objects contain one or more integration components, which in turn contain one or more Integration Fields. For details on integration objects, see *Platform Technologies: Siebel eBusiness Application Integration Volume II, MidMarket Edition*.

Figure 9 illustrates the Siebel Data Mapping architecture.



**Figure 9. The Siebel Data Mapper and the EAI Data Mapping Engine Architecture**

A data map defines the relationship between *source* and *target* object format. The map controls the transformation process. Transformation maps are stored in the Siebel database as explained in [Table 8](#).

**Table 8. Maps and Data Table Relationship**

| Map Type                          | Siebel Data Table |
|-----------------------------------|-------------------|
| Integration Object maps           | S_INT_OBJMAP      |
| Integration Object Component maps | S_INT_COMPMAP     |
| Integration Object Field maps     | S_INT_FLDMAP      |

## Integration Object Maps

An integration object map is the top-level data map specifying mapping from one integration object to another. An integration object map contains one or more integration component maps and can optionally contain integration map arguments.

### Integration Map Arguments

Data maps can be parameterized using integration map arguments. Map arguments can be referenced in any expression, including the integration field map expression, Source Spec expression, precondition expression, and post condition expressions. For example, you may want to have a field map that creates an Order Number in the target object by prefixing the Order Number in the source object with a constant.

You may want to use this map for orders coming from multiple partners and use a different prefix for each partner. To achieve this with a single data map, you can define an argument Prefix in the Integration Map Argument List, and use this argument Prefix in the field map source expression: [&Prefix] + [Order Number]. Then in the input method arguments in EAI Data Mapping Engine business service, you can specify any value for Prefix.

### Integration Component Maps

Integration component maps specify how integration components in the source object get mapped to integration objects in the target object. For every occurrence of the source component in the source integration object instance, an instance of the target component is created in the target object instance. An integration component map contains one or more integration field maps. For details on integration component maps, see [“Creating Integration Component Maps” on page 68](#).

### Integration Field Maps

Integration field maps specify how fields in the source integration object are mapped to fields in the target integration component. An integration field map target is always a field in the target component of the parent component. An integration field map source can be a constant, a reference to a map argument, a field in the source component, or other legally addressable components such as ancestors of the source component. It can also be a Siebel Query Language expression using one or more of the preceding elements.

---

**NOTE:** For details on integration field maps, see [“Creating Integration Field Maps” on page 69](#). For details on addressing fields in components other than the source component, see [“Addressing Fields in Components” on page 84](#). For details on Source Expression, see [“Source Expressions” on page 80](#).

---



## Creating Data Maps

The following checklist provides the high-level steps for creating data maps.

### Checklist

- 
- ☐ Define and validate integration objects and determine the required maps.  
For details, see [“Define Integration Objects.”](#)

---

  - ☐ List components and fields within the Siebel object to use.  
For details, see [“Define Integration Objects.”](#)

---

  - ☐ Create a map between the two integration objects.  
For details, see [“Creating New Data Maps” on page 66.](#)

---

  - ☐ Create maps between the components of the objects you mapped.  
For details, see [“Creating Integration Component Maps” on page 68.](#)

---

  - ☐ Create maps between individual fields within the components you mapped.  
For details, see [“Creating Integration Field Maps” on page 69.](#)

---

  - ☐ Validate the data maps.  
For details, see [“Validating the Data Map” on page 69.](#)
- 

### Define Integration Objects

Before you create a data map, you need to verify that valid integration objects exist for the source and the target data you want to map. For details on creating and validating integration objects, see *Platform Technologies: Siebel eBusiness Application Integration Volume II, MidMarket Edition*.

## Determining Required Maps

The Integration Object Browser lists the existing integration object maps. Use this browser to determine which maps you need to create.

### **To determine which maps to create**

- 1 From the application-level menu, choose View > Site Map > Integration Administration > Data Map Browser.

The Integration Object Map appears.

- 2 Query for the integration objects you want to map.

## Creating New Data Maps

Once you determine what objects you need to map, use the Data Map form to create data maps. See [“Define Integration Objects” on page 65](#).

### **To create a new data map**

- 1 From the application-level menu, choose View > Site Map > Integration Administration > Data Map.

The Integration Object Map list appears.

- 2 In the Integration Object Map list, click New to create a new map.

- 3 Provide the necessary fields:

**Name.** Enter a name for the map you are creating.

**Source Object Name.** From the list of values, select the source integration object you want to create the data mapping for.

**Target Object Name.** From the list of values, select the target integration object into which you want the data to be transferred.

- 4 Click Save.

## Creating Maps Using Auto-Map

Once you have created your integration object map, you can use the Auto-Map button to have the Siebel application create the necessary mappings between the underlying components. The root components are always mapped by Auto-Map, whether they have the same name or not. Once the root components are mapped, the Auto-Map will recursively walk through all components and their fields to map them. If the components have the same name, the Auto-Map continues to map their fields and their children components. However, if the components have different names, the Auto-Map ignores the current components, their fields, and their children components, and moves on to map the next component. In cases where only the field names are different, the Auto-Map only ignores that one field and continues with its recursive mapping.

---

**NOTE:** You can also use the Auto-Map on an existing mapping when you modify the integration object. The Auto-Map does not overwrite your manual mappings.

---

## Defining Arguments for a Data Map (Optional)

After you create a data map, you can define the arguments for your map. You can then use these arguments when you call the map within workflow. To define arguments, use the Integration Map Argument list on the Integration Object Map form.

### *To define integration map arguments*

- 1** Create a new record in the Integration Map Argument list.
- 2** Provide the following fields:
  - Name.** Enter a name for the argument.
  - Data Type.** From the list of values, select the Siebel Data Type for the argument.
  - Display Name.** Enter the name that you want displayed.
- 3** Click Save.

## Creating Integration Component Maps

Once you have defined a data map (see [“Creating New Data Maps” on page 66](#)), you need to set up the mapping between the components and the fields within the objects you have mapped. You do this using the Data Map Editor form. The Integration Object Editor list displays existing object maps and provides views in which you can define maps for components and for fields. You use the Integration Component Map view to create integration component maps.

### **To define integration component maps**

- 1 From the application-level menu, choose View > Site Map > Integration Administration > Data Map Editor.
- 2 In the Integration Object Map list, select the map for which you want to define integration component maps.
- 3 Create a new record in the Integration Component Map list.
- 4 Provide the following fields.

**Name.** Name of the map you are creating.

**Source Component Name.** The component where you are getting the data.

**Target Component Name.** The component where you want to store the data.

**Source Search Specification (optional).** The search criteria based on which the records are filtered. See [“Source Search Specifications” on page 81](#) for details.

**Parent Component Map Name (optional).** The parent component field is used when there is a mapping to two target components that share multiple parent components. You can exclude data from one of these child objects by choosing a parent component.

**Precondition (optional).** See [“Preconditions” on page 82](#) for details.

**Postcondition (optional).** See [“Postconditions” on page 83](#) for details.

- 5 Click Save.

## Creating Integration Field Maps

You define the integration field map between your source and target fields using the Integration Field Map form.

### **To define a integration field map**

**1** Create a new record in the Integration Field Map list.

**2** Provide the following fields:

**Target Field Name.** Name of the field in the Target Component where the value will be assigned.

**Source Expression.** An expression that is used to calculate a value for the Destination Field. See [“Source Expressions” on page 80](#) for details.

**3** Click Save.

## Validating the Data Map

Once you have created your data map, you need to validate your data map.

### **To validate your data map**

**1** From the application-level menu, choose View > Site Map > Integration Administration > Data Maps.

**2** Select your data map.

**3** Click Validate to validate your data map.

**4** Take necessary action to fix the problems with your map or the associated integration objects.

# Examples of Workflow Process

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

## Outbound Workflow Process

To execute the map for an outbound process create a workflow process to query the database, purge the data map, execute the data map, and then write the XML into a file. The following examples illustrate integration between contact and employee business objects.

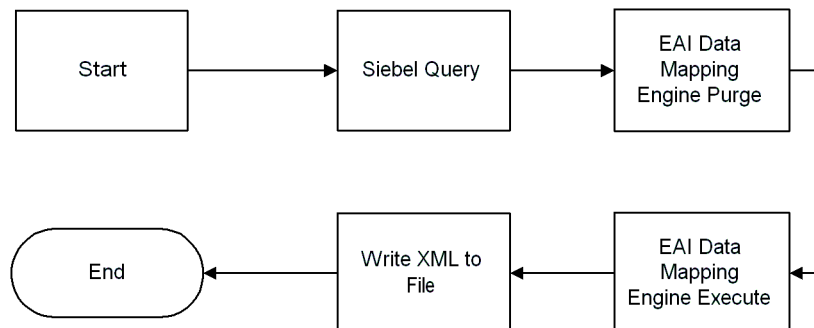
### To create an outbound workflow process

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

---

**NOTE:** The EAI Data Mapping Engine Purge step should only be used in a development environment.

---



- 3** Define the process properties, using the following table as a guide.

| <b>Input Argument</b>      | <b>Type</b> |
|----------------------------|-------------|
| Contact Message            | Hierarchy   |
| DTE XML                    | Hierarchy   |
| Process Instance Id        | String      |
| Error Code                 | String      |
| Error Message              | String      |
| Object Id                  | String      |
| Siebel Operation Object Id | String      |

- 4 The second business service queries the information from the database using the EAI Siebel Adapter business service with Query method.

| Field            | Value              |
|------------------|--------------------|
| Name             | Siebel Query       |
| Business Service | EAI Siebel Adapter |
| Method           | Query              |

| Input Argument                 | Type    | Value                                |
|--------------------------------|---------|--------------------------------------|
| Output Integration Object Name | literal | An Employee                          |
| Search Specification           | Literal | [Employee.Last Name] LIKE "Peterson" |

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

**NOTE:** For more information on using the EAI Siebel Adapter, see *Platform Technologies: Siebel eBusiness Application Integration Volume II, MidMarket Edition*.

- 5 The third business service purges the map using the Data Mapping Engine business service with the Purge method. This step is only for development mode so that the latest map is picked for the process and should not be used in a production environment. This step does not require any input or output arguments.

| Field            | Value                   |
|------------------|-------------------------|
| Name             | DDTE Purge              |
| Business Service | EAI Data Mapping Engine |
| Method           | Purge                   |



- 6 The forth business service executes the data map using the EAI Data Mapping Engine business service with the Execute method.

| Field            | Value                   |
|------------------|-------------------------|
| Name             | DDTE Execute            |
| Business Service | EAI Data Mapping Engine |
| Method           | Execute                 |

| Input Argument                 | Type             | Value             | Property Name    | Property Data Type |
|--------------------------------|------------------|-------------------|------------------|--------------------|
| Map Name                       | Literal          | Outbound DDTE Map |                  |                    |
| Output Integration Object Name | Literal          | My DTE            |                  |                    |
| Siebel Message                 | Process Property | -                 | Employee Message | Hierarchy          |

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

- The last business service writes the XML into a file using the EAI XML Write to File business service with the Write Siebel Message method.

| Field            | Value                 |
|------------------|-----------------------|
| Name             | Write XML to File     |
| Business Service | EAI XML Write to File |
| Method           | Write Siebel Message  |

This step requires the following input argument.

| Input Argument | Type             | Value      | Property Name | Property Data Type |
|----------------|------------------|------------|---------------|--------------------|
| File Name      | Literal          | c:\emp.xml |               |                    |
| Siebel Message | Process Property | -          | Int.ObjName   | Hierarchy          |

The output argument for this step is optional and can be defined as follows.

| Property Name | Type            | Value | Output Argument |
|---------------|-----------------|-------|-----------------|
| IntObjName    | Output Argument | -     | Siebel Message  |

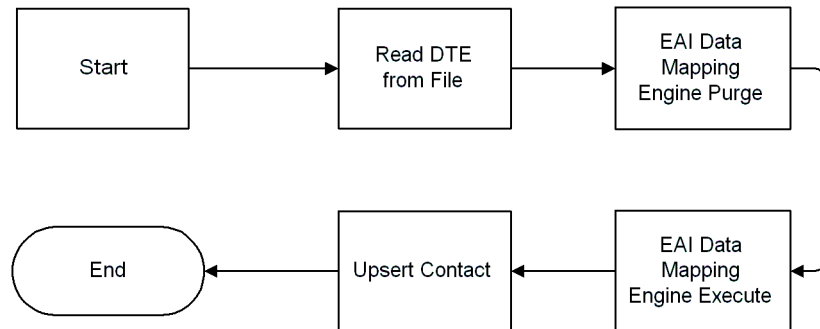
## Inbound Workflow Process

To execute the map for an inbound process you need to create a workflow process to read the data from a file, purge the data map, execute the data map, and then write the XML into a file.

### To create an inbound workflow process

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

**NOTE:** The DDTE Purge step should only be used in a development environment.



- 3 Define the process properties using the following table as a guide.

| Name                       | Data Type |
|----------------------------|-----------|
| Employee Message           | Hierarchy |
| IntObjName                 | Hierarchy |
| Process Instance Id        | String    |
| Error Code                 | String    |
| Error Message              | String    |
| Object Id                  | String    |
| Siebel Operation Object Id | String    |

- 4 The first business service reads the information from a file using the EAI XML Read from File business service with the Read Siebel Message method.

| Field            | Value                  |
|------------------|------------------------|
| Name             | Read QA DTE from File  |
| Business Service | EAI XML Read from File |
| Method           | Read Siebel Message    |

| Input Argument | Type    | Value      |
|----------------|---------|------------|
| File Name      | Literal | c:\emp.xml |

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

- 5 The second business service purges the map using the Data Mapping Engine business service with the Purge method. This step is only for development mode so that the latest map is picked for the process and should not be used in a production environment.

| Field            | Value                   |
|------------------|-------------------------|
| Name             | DDTE Purge              |
| Business Service | EAI Data Mapping Engine |
| Method           | Purge                   |

This step does not require any input or output arguments.

- 6 The third business service executes the data map using the EAI Data Mapping Engine business service with the Execute method.

| Field            | Value                   |
|------------------|-------------------------|
| Name             | DDTE Execute            |
| Business Service | EAI Data Mapping Engine |
| Method           | Execute                 |

| Input Argument                 | Type             | Value            | Property Name | Property Data Type |
|--------------------------------|------------------|------------------|---------------|--------------------|
| Map Name                       | Literal          | Inbound DDTE Map |               |                    |
| Output Integration Object Name | Literal          | A Contact        |               |                    |
| Siebel Message                 | Process Property | -                | DTE Message   | Hierarchy          |

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

- 7 The last business service writes the data into the database using the EAI Siebel Adapter business service with Insert or Update method.

| Field            | Value              |
|------------------|--------------------|
| Name             | Upsert Contact     |
| Business Service | EAI Siebel Adapter |
| Method           | Insert or Update   |

This step requires the following input argument.

| Input Argument | Type             | Property Name   | Property Data Type |
|----------------|------------------|-----------------|--------------------|
| Siebel Message | Process Property | Contact Message | Hierarchy          |

This step does not require output parameters.

---

**NOTE:** For more information on using the EAI Siebel Adapter, see *Platform Technologies: Siebel eBusiness Application Integration Volume II, MidMarket Edition*.

---

- 8 Use the Workflow Simulator to run through the steps you created in your workflow process to test your process.

---

**NOTE:** For details on creating a workflow process and using the Workflow Simulator to test your workflow process, see *Siebel Business Process Designer Administration Guide, MidMarket Edition*.

---

## Executing the Workflow

Once you have designed and tested your workflow, you can run it in your production using Workflow Process Manager Server.

---

**NOTE:** For more details on how to activate and execute a workflow, see *Siebel Business Process Designer Administration Guide, MidMarket Edition*.

---

## EAI Data Mapping Engine Expressions

The EAI Data Mapping Engine uses four categories of expressions:

- Source expressions
- Source search specifications
- Preconditions
- Postconditions

These expressions support Siebel Query Language expressions. These expressions can address fields in the source component, map arguments, and constants. In addition to fields in the source component, fields in certain other components in the source integration object can be addressed. For details, see [“Addressing Fields in Components” on page 84](#). These expressions are just like Siebel Query Language support invocations of predefined functions and custom business services.

---

**NOTE:** For details on the Siebel Query Language, see *Siebel Tools Online Help, MidMarket Edition*.

---

### Source Expressions

Source Expressions is a required field for every integration field map. The source expression can be a literal or based on scripting if you need to parse data, or if you need to query the database for a specific value. The source expression is associated with an instance of the input integration component named in the integration component map, which is the parent of the integration field map that contains the source expression. An example of a source expression is:

```
[First Name] + " " + [Last Name]
```



This expression concatenates the First Name and the Last Name and separates them with a space to be moved into a target field such as Full Name.

---

**NOTE:** Only a subset of Siebel Query Language Expressions that do not require context of a business component, is supported by EAI Data Mapping Engine. You can not use the following Siebel Query Language Expressions that require context of a business component in the Source Expression: BCName (), Count (mvlink), IsPrimary (), Min (mvfield), Max (mvfield), ParentBCName (), ParentFieldValue (field\_name), Sum (mvfield), GetXVal (), GetXValAsNum (), GetXValAsInt (), GetXValAsDate (), and XAIsClass ().

---

## **Source Search Specifications**

Source Search Specification is a Boolean expression that is used to determine if a given component instance satisfies given criteria. It may only appear in an integration object map or a integration component map together with an integration component name. Defining a Search Specification is optional, and if you do not define it then it does not apply any criteria and returns True.

If a field in the current integration component has the same name as a field in a parent component then you can only address the parent component field by using dot ('.') notation. An example of a Source Component Search Specification is:

```
[Role] = "Billing"
```

The expression returns True only if the current input integration component has the value Billing in the Role field.

---

**NOTE:** If no Search Specification is provided, then every input integration component whose type matches the input component of the integration component map is processed.

---

### Preconditions

You can use preconditions to make sure that a field of the input object has a certain value or otherwise terminate the process. An error is generated if the field in the input object has any other value, or no value. Preconditions are evaluated immediately before their containing integration component map is executed. If the condition is true then the process continues. If the condition is false then the whole transformation is aborted and EAI Data Mapping Engine returns an error to the caller. An example of a precondition is:

```
[Role]="Billing" Or [Role]="Shipping"
```

This precondition makes sure that the field Role of the input object either has a value Billing or a value Shipping before it proceeds with the process of data transformation.

The precondition is only applied to the input components that are selected by the Source Search Specification. The input components that fail to match the Source Search Specification will not be checked against the precondition.

A precondition expression may address any field in the current input component, and any of its parent components. It can also address any service call parameter that has been declared as a map argument.

---

**NOTE:** The default value for the precondition is True. If the precondition is omitted from an integration component map then no constraint is enforced.

---

## Postconditions

Postconditions are evaluated and applied to the newly created objects when you execute the containing integration component map. If the result of the postcondition is true then the process continues. If the result is false, the whole transformation is aborted and EAI Data Mapping Engine returns an error. Here is an example of a postcondition:

```
[Object ID]<>" " Or ([First Name]<>" " And [Last Name]<>" ")
```

This postcondition checks the output component for a value in the Object ID or in both the First Name and the Last Name.

---

**NOTE:** Since there is no search specification for output components, the postcondition is applied only once for every output component instantiated because it executes its containing integration component map.

---

The type of the expression may be any type that can be assigned to the Destination Field type either directly or after applying standard conversions to the result of the expression.

## Addressing Fields in Components

You may want to address fields in components other than the source component because your target component may depend on more than one component in the source object. In such cases, you cannot use different component maps with different source components and the same target component because each component map creates a different instance of the target component. Data Mapping Engine Expressions allow addressing fields in source integration object components other than the source component using the dot notation, [Component Name.Field Name].

---

**NOTE:** The picklist for the source expression in the Data Mapper View does not list fields in components other than the source component. Such fields should be typed in using the dot notation.

---

Addressing fields in other components is legal only if the cardinality of the component is less than or equal to one relative to the source component—that is, only if the component can be uniquely identified from the context of the source component without using any qualifiers other than the component name. If a field in a component that is not legally addressable is used in the source expression then it leads to a runtime error to the effect that such a field does not exist. Any component that is an ancestor of the source component in the integration object hierarchy has a relative cardinality of 1 which means it can always be uniquely identified from the source component. Therefore, fields in ancestor components can always be legally addressed.

Sibling components can be uniquely identified from the context of the source component only if they do not occur multiple times—that is have a cardinality of less than or equal to 1. Only such siblings can be legally addressed. Therefore, it is not legal to address repeated sibling components. Components that are descendants of a sibling component can be legally addressed only if there is no repeated component in the hierarchical path from the sibling component to the component.

Further, components that are descendants of a sibling of some ancestor of a source component can be legally addressed only if there is no multiply-occurring component in the hierarchical path from the sibling-of-ancestor-of-source component to the component.

## **Data Mapping Scenario**

The scenarios below concern an IT developer named Chris Conway, who works for a computing company, PCS Computing. One of his responsibilities is creating and maintaining the data mappings between Siebel and the other applications in use at PCS. The first task he is assigned to is to create mapping between Siebel applications and the external application they need to integrate with.

### **Mapping Between Siebel and an External Application**

Chris is in charge of integrating PCS's Siebel implementation with a custom in-house application. The purpose is to exchange customer information between the two systems.

After weighing various options, Chris decides to use the Siebel Data Mapper instead of scripts to perform the data mapping. He creates the Internal Integration Object using the Siebel Integration Object Wizard from within Siebel Tools. He also creates an external integration object using the external application's DTD.

When Chris is ready to map the two integration objects, he navigates to the Data Mapper and creates a new entry by supplying the name of the map and associating the internal integration object with the external integration object, as explained in [“Creating New Data Maps” on page 66](#). He then uses the Map Editor form to create object, component, and field maps, as explained in [“Creating Integration Component Maps” on page 68](#).

When he finishes creating the map, Chris navigates to the Siebel Workflow Process Designer form to define the integration flow. For one of the workflow steps, he defines an invocation of the Siebel Data Mapper. He supplies the appropriate parameters, including the name of the map, and saves his work.



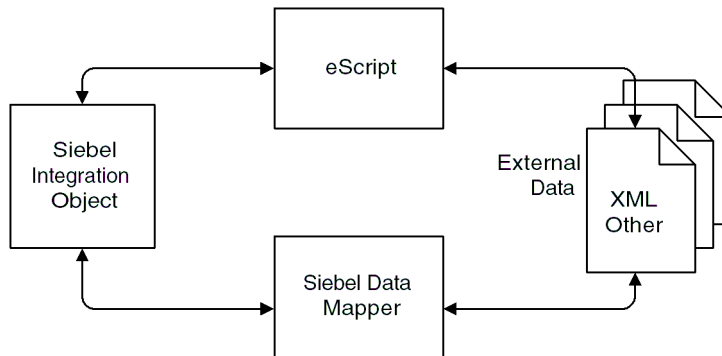
## **Data Mapping Using Scripts**

# **4**

This chapter describes the process of using the Siebel eScript Data Mapping to convert your external data to the Siebel format and your Siebel data to your external data specifications.

## Overview

You can accomplish your data transformation requirements in Siebel eBusiness applications by using the Data Transformation Function or Siebel Data Mapper, as illustrated in [Figure 10](#).



**Figure 10. Data Transformation Options**

For customers who want to do data mapping within Siebel applications, Siebel applications now support two data mapping solutions—Siebel Data Mapper and Siebel eScript Data Mapping. Siebel Data Mapper has a declarative interface and requires no programming skills. Siebel eScript Data Mapping uses scripts programmed in eScript as data maps.

Data maps defined using Siebel Data Mapper are easy to maintain and upgrade. These maps also perform better than eScript Data Maps. Since Siebel Data Mapper is based on a declarative interface, it does not have the full flexibility and power that the data mapping using eScript has. Siebel Data Mapper should suffice for most integration needs except some complex mapping situations requiring aggregation, joins, or programmatic flow control.



The following checklist outlines the main steps required to accomplish your data transformation requirements using the Data Transformation Functions.

## Checklist

- ☐ Define integrationt objects in Siebel Tools—one to represent Siebel objects and another to represent external data.  
For details, see *Platform Technologies: Siebel eBusiness Application Integration Volume II, MidMarket Edition*.
- ☐ Set up the Data Transformation Map.  
For details, see [“Setting Up a Data Transformation Map” on page 90](#).
- ☐ Write the Siebel eScript code to perform the data transformation.  
For details, see [“To write a script for DTE business service” on page 91](#).

# EAI Data Transformation

The Siebel Data Transformation Functions are a framework for building data transformation maps. Data transformation maps act as import and export filters, preparing data from an external system for entry into Siebel applications and preparing data in Siebel applications for export.

Data transformation maps are created as business services using Siebel eScript. You invoke them as part of an eAI Workflow process.

A data transformation map reads data from an input structure and transfers it to an output structure, transforming it along the way. The map developer creates a custom eScript function to do the transformation. The Data Transformation Functions provide a convenient way to read the input data and generate results. They also provide a framework for invoking your map functions, handling errors, and accessing other eAI resources.

## Setting Up a Data Transformation Map

You create your data transformation map in Siebel Tools in a business service, then you compile it into an .srf file. You can organize your maps in many different ways. Each business service you create can contain one or more maps. You can, in fact, use several business services to organize a large number of maps into logical groups.

### ***To define a data transformation business service in Siebel Tools***

- 1** Run Siebel Tools.
- 2** Choose a locked project.
- 3** Create a new business service.
- 4** Choose the *CSSEAITEScriptService* class for the business service.

- 5 Double-click the Business Services Methods folder and add the method *Execute*.

Select the Business Service Method Arg folder and add the arguments for the Execute method. For a list of arguments and their description, see [“DTE Business Service Method Arguments” on page 93](#). The arguments to include are:

- MapName
- An input argument. Select one of SiebelMessage, XMLHierarchy, or MIMEHierarchy as the argument name, based on the type of input.
- An output argument. Required if the output object is a different type than the input argument. Select one of SiebelMessage, XMLHierarchy, or MIMEHierarchy as the argument name.
- If the input and output types are the same then the same argument entry is used for both.
- OutputType
- InputType (Optional). This is required only when passing the business service input property set to the map function without interpretation. This is done by specifying the InputType as ServiceArguments.

---

**NOTE:** Most transform maps use SiebelMessage for both the input and output arguments. This is for mapping one integration object to another. For details, see [“DTE Business Service Method Arguments” on page 93](#).

---

Once you have created the business service you need to write the Siebel eScript code to perform the data transformation.

**To write a script for DTE business service**

- 1 Choose the Business Service object and select the business service you want to contain the transformation map.
- 2 Right-click to display the pop-up menu.
- 3 Choose Edit Server Scripts and choose eScripts as the scripting language if you are prompted to select scripting language.

- 4 In the (declarations) procedure of the (general) object, add the line:

```
#include "eaisiebel.js"
```

- 5 In the *Service\_PreInvokeMethod* function of the service, change the function to the following:

```
function Service_PreInvokeMethod (MethodName, Inputs, Outputs)
{
    return EAIExecuteMap (MethodName, Inputs, Outputs);
}
```

Your data transformation map is run as a business service invoked from a Workflow process. Business service scripts have a standard entry point, *Service\_PreInvokeMethod*. Although the script environment provides you with a boilerplate function by this name, you need to modify it, as described in the preceding steps, to include the call to the *EAIExecuteMap* function.

- a The *MethodName* must be *Execute* and is used by Siebel Workflow. The name of your function is the name you supply for the *MapName* argument to the *Execute* method.
- b *Inputs* is the input message from Workflow containing service arguments—for example, *MapName* and *Output Integration Object Name*—and the integration message to be transformed. *Outputs* is the argument used to return data—for example, *Siebel Message*. *MapName* specifies the map function to be executed and must be the name of one of the functions you defined in the business service.

## DTE Business Service Method Arguments

Table 9 lists the arguments for the Execute method of the DTE business services.

**Table 9. DTE Business Service Method Arguments**

| Name & Display Name             | Data Type | Type         | Optional | Storage Type | PickField | PickList                     |
|---------------------------------|-----------|--------------|----------|--------------|-----------|------------------------------|
| MapName<br>Map Name             | String    | Input        | No       | Property     |           |                              |
| InputType<br>Input Type         | String    | Input        | No       | Property     | Value     | EAI Message Type<br>PickList |
| OutputType<br>Output Type       | String    | Input        | No       | Property     | Value     | EAI Message Type<br>PickList |
| SiebelMessage<br>Siebel Message | Hierarchy | Input/Output | Yes      | Hierarchy    |           |                              |
| MIMEHierarchy<br>MIME Hierarchy | Hierarchy | Input/Output | Yes      | Hierarchy    |           |                              |
| XMLHierarchy<br>XML Hierarchy   | Hierarchy | Input/Output | Yes      | Hierarchy    |           |                              |

You can set these arguments in Siebel Tools.

**MapName.** The name of the eScript function to call to perform the transformation.

**InputType.** The type of input object to pass to the transformation function. The value will be one of SiebelMessage, MIMEHierarchy, XMLHierarchy, or ServiceArguments. This argument is required only when you use ServiceArguments as the value. When ServiceArguments is used the business service, PropertySet is passed to the map function without interpretation.

**OutputType.** The type of the output object to pass to the transformation function. The types are the same as the ones for Input Type.

**SiebelMessage.** You use this argument when the input *and* or *or* output object is a SiebelMessage. SiebelMessage is used when converting to or from an integration object. SiebelMessage is the correct choice when mapping one integration object to another. Your map function is passed an object of type CSSEAIIntMsgIn for the SiebelMessage that is the input to the transformation and an object of type CSSEAIIntMsgOut for the SiebelMessage that is produced by the transformation.

**MIMEHierarchy.** You use this argument if the input *and* or *or* output object is a MIMEHierarchy. MIMEHierarchy is used when converting to or from MIME Hierarchy objects. Your map function is passed two object types; CSSEAIMineMsgIn for the MIMEHierarchy that is the input to the transformation and CSSEAIMimeMsgOut for the MIMEHierarchy that is produced by the transformation. MIME Hierarchy objects are defined by the EAI MIME Doc Converter business service. For details on the EAI MIME Doc Converter, see *Platform Technologies: Siebel eBusiness Application Integration Volume II, MidMarket Edition*.

**XMLHierarchy.** You use this argument if the input *and* or *or* output object is an XMLHierarchy. XMLHierarchy is used when converting to or from XML Hierarchy objects. Your map function is passed an object of type XML Property Set for both input and output XMLHierarchy. XML Hierarchy objects are defined by the XML Hierarchy Converter business service. For details on XML Hierarchy Converter, see *XML Reference: Siebel eBusiness Application Integration Volume V, MidMarket Edition*.

## Map Functions

A map function has the following signature:

```
function MapFnName (objectIn, objectOut)
```

The function name signified by *MapFnName* is the name of your transformation function. It is the value passed as the MapName argument to the business service. The Input Type and Output Type business service arguments determine the types of the *objectIn* and *objectOut* arguments and default to the type *Integration Message*. You should name these arguments according to type. For example, to use the default values, you would specify a function that transforms one integration object to another as:

```
function MapFnName (intMsgIn, intMsgOut)
```

If you define a function that transforms an XML property set to an integration object, you might specify it as:

```
function MapFnName (xmlPropSetIn, intMsgOut)
```

The arguments to these functions are contained within the input and output arguments to the business service's *Service\_PreInvokeMethod* function. The *EAIExecuteMap* function—called by *Service\_PreInvokeMethod*—interprets the arguments and passes them to *MapFnName*. *MapFnName* reads from the input object and writes to the output object using the appropriate API for each type of object.

If you define a function to access input integration object, you might specify it as:

```
Function myMapFn (ObjectIn, ObjectOut) {
  inIntObj = ObjectIn.GetIntObj(); //Get Integration Object
  //Iterate over all Integration Object Instances
  While (inIntObj.NextInstance()) {
    //Get the Primary Component which is called "Order Entry - Orders"
    primaryIntComp = inIntObj.GetPrimaryIntComp("Order Entry - Orders");
    //Iterate over all instances of Primary Component
    while (primaryIntComp.NextRecord()) {
      OrderId = primaryIntComp.GetFieldValue ("Id");
      //Get component "Order Entry - Line Items" which is child of
      "Order Entry - Orders"
      comp = primaryIntComp.GetIntComp ("Order Entry - Line Items");
      //Process component similar to primary component
      while (comp.NextRecord()) {
```

```
OrderItemId = comp. GetFieldValue ("Id");
And to define a function to create output integration object, you
might specify it as:
Function myMapFn (ObjectIn, ObjectOut) {
outIntObj = ObjectOut.CreateIntObj("Sample Order");
While (Need new integration object instances) {
    outIntObj.NewInstance();
    //Create Primary Component which is called "Order Entry -
Orders"
primaryIntComp = inIntObj.CreatePrimaryIntComp("Order Entry -
Orders");
while (Need new instances of primary int component) {
    primaryIntComp.NewRecord();
    primaryIntComp.SetFieldValue ("Id", OrdertemId);
    //Create component Order Item which is child of Order
comp = primaryIntComp.CreateIntComp ("Order Entry - Order
Items");
    //Process component similar to primary component
    while (need new instances of component) {
        comp.NewRecord();
        comp. SetFieldValue ("Id", OrdertemId);
```

## **EAIExecuteMap() Method**

This method executes a user-defined data transformation function. [Table 10](#) lists the parameters for this method.

**Syntax**    `EAIExecuteMap(methodName, inputPropSet, outputPropSet)`

**Table 10.    Parameters for EAIExecuteMap() Method**

| Parameter     | Description  |
|---------------|--|
| methodName    | The business service method should be <i>Execute</i> . |
| inputPropSet  | Input message and service arguments.                   |
| outputPropSet | Output message and service arguments.                  |

**Returns**    *CancelOperation* or *ContinueOperation*. The *Service\_PreInvokeMethod* function should return the value returned by the *EAIExecuteMap*.

**Usage**    See [“Setting Up a Data Transformation Map” on page 90](#).



## The Data Transformation Functions

The data transformation API consists of global functions and classes that represent the different parts of input and output data. The data transformation functions are implemented as Siebel eScript. You must use Siebel eScript to create your data transformation maps.

Three different top-level data types are supported:

- **Siebel Messages.** See [“Siebel Message Objects and Methods” on page 98.](#)
- **MIME Messages.** See [“MIME Message Objects and Methods” on page 120.](#)
- **XML Property Sets.** See [“XML Property Set Functions” on page 129.](#)

The data type is determined by the `InputType` and `OutputType` arguments, as described in [“DTE Business Service Method Arguments” on page 93.](#)

Siebel Messages are the most common data type. Siebel Messages are a hierarchical type represented at the top level by an Integration Message message. See [“Siebel Message Objects and Methods” on page 98.](#)

It is also possible to operate directly on the business service input and output property sets. This is accomplished by specifying the `InputType` or `OutputType` as `ServiceArguments`. In this case the business service property set arguments are passed directly to the map function. The standard property set functions can be used to access them.

# Siebel Message Objects and Methods

A Siebel Message is a message containing the data of individual integration object instances. It is hierarchically structured and composed of several different types of objects.

The data transform API uses several different eScript classes to represent a Siebel Message:

- **An integration message.** This represents the top-level message container. See [“Integration Message Objects” on page 98](#).
- **An integration object.** See [“Integration Object Objects” on page 104](#).
- **A primary integration component.** See [“Primary Integration Component Objects” on page 107](#).
- **Integration components.** See [“Integration Component Objects” on page 113](#).

Each of these parts of a Siebel Message has two classes: one for input and one for output. Each class provides methods for specific purposes.

## Integration Message Objects

The integration message is the top-level piece of a *message*. The workflow process passes the integration message to the Data Mapping Engine as input. The Data Mapping Engine returns another message as output. The integration message object provides access to workflow arguments, integration message arguments, and the integration object that is contained in the message.

The following integration message objects are provided:

- CSSEAIIntMsgIn
- CSSEAIIntMsgOut

## CSSEAllIntMsgIn

This object represents an integration message that is open for reading. The object provides `GetArgument` and `GetIntObj` methods.

### GetArgument() Method

This method gets the value of a business service argument. For example, this could get the name of a map function in the business service. [Table 11](#) lists the parameters for this method.

**Syntax** `GetArgument(name [, defaultIfNull [, defaultIfEmpty]])`

**Table 11. Parameters for GetArgument() Method**

| Parameter      | Description  |
|----------------|--|
| name           | The name of a business service argument.                             |
| defaultIfNull  | Returned if a service argument of the specified name does not exist. |
| defaultIfEmpty | Returned if the service argument is set to an empty string.          |

**Returns** String or null.

**Usage** Use this method to get the value of an argument passed to the business service. For example, if the `MapName` argument passed to the business service is `MapExtOrderToOrder`, the call:

```
intMsgIn.GetArgument ( "MapName" );
```

returns the name of the map, `MapExtOrderToOrder`, passed to the business service.

If the named argument does not exist, `null` is returned. If the named argument exists but the value is the empty string, the empty string is returned. You can use the `defaultIfNull` and `defaultIfEmpty` optional arguments to change this behavior.

The arguments `defaultIfNull` and `defaultIfEmpty` are optional; however, if you specify `defaultIfEmpty`, you must also specify the `defaultIfNull` argument.

**GetIntObj() Method**

This method returns an instance of the integration object and opens it for reading. [Table 12](#) lists the parameter for this method.

**Syntax**     GetIntObj(*name*)

**Table 12.    Parameter for GetIntObj() Method**

| Parameter | Description  |
|-----------|--|
| name      | The name of an integration object in the active integration message. |

**Returns**     CSSEAllIntObjIn Integration Object

**Usage**        An integration object instance is always returned even if the integration object does not exist. Call the returned object’s Exist method to test for this before calling other methods on the object. An error is raised if an integration object is present but the name is not correct.

---

**NOTE:** Currently an integration message can contain only one integration object.

---

**GetAttachmentCount () Method**

This method returns the number of attachments in the input integration message.

**Syntax**        GetAttachmentcount()

**Returns**        The number of attachments in the input integration message.

## GetAttachment () Method

This method returns the attachment specified by the index. [Table 13](#) lists the parameter for this method.

**Syntax**     GetAttachment(index)

**Table 13. Parameters for GetAttachment() Method**

| Parameter | Description                            |
|-----------|--|
| index     | The index of the attachment to return. |

**Returns**     The attachment (a PropertySet) specified by the index. The index is zero based. Returns null if index is out of bounds.

## GetAttachmentByCID () Method

This method retrieves an attachment based on the Content Identifier (CID). [Table 14](#) lists the parameter for this method.

**Syntax**     GetAttachmentByCID(cid)

**Table 14. Parameters for GetAttachmentByCID() Method**

| Parameter | Description                               |
|-----------|---|
| cid       | The Content Identifier of the attachment. |

**Returns**     The attachment (a PropertySet) specified by the CID. Returns null if there is no attachment with the specified CID.

# CSSEAllIntMsgOut

This object represents an output integration message that is open for writing. The object provides CreateIntObj and SetArgument methods:

## CreateIntObj() Method

This method creates a new integration object. [Table 15](#) lists the parameter for this method.

**Syntax**      CreateIntObj(*name*)

**Table 15.    Parameter for CreatIntObj() Method**

| Parameter | Description  |
|-----------|--|
| name      | Creates a new integration object and adds it to the integration message. |

**Returns**      CSSEAllIntObjOut Output Integration Object

**Usage**      An integration message can contain only one integration object, so multiple calls to this method on one integration message raises an error. The name must agree with the business service argument *OutputIntObjName*, if that argument is passed to the service.

## SetArgument() Method

This method sets the value of a business service argument. [Table 16](#) lists the parameters for this method.

**Syntax**      SetArgument(*name, value*)

**Table 16.    Parameters for SetArgument() Method**

| Parameter | Description  |
|-----------|--|
| name      | The name of an argument in the active business service.                            |
| value     | The string value corresponding to the argument named by the <i>name</i> parameter. |

**Returns** Not applicable

**Usage** You can call the *SetArgument* method to establish the value of a given output argument for the business service method invocation.

### SetAttachmentSource () Method

This method establishes the source object to copy attachment objects from. The source object must be a *CSSEAIIntMsgIn*, *CSSEAIMimeMsgIn*, or other object implementing the *GetAttachmentByCID* method. [Table 17](#) lists the parameter for this method.

**Syntax** SetAttachmentSource(source)

**Table 17. Parameters for SetAttachmentSource() Method**

| Parameter | Description            |
|-----------|------------------------|
| source    | The attachment source. |

### CopyAttachment (cid) Method

This method copies an attachment from the attachment source to the output integration object. The attachment is referenced by the MIME Content Identifier (CID). The attachment source must be established by calling *CSSEAIIntMsgOut.SetAttachmentSource* prior to calling this method. [Table 18](#) lists the parameter for this method.

**Syntax** CopyAttachment(cid)

**Table 18. Parameters for CopyAttachment() Method**

| Parameter | Description              |
|-----------|--------------------------|
| cid       | MIME content identifier. |

**Returns** The attachment copy is returned as a property set. This method returns null if the attachment source does not contain an attachment with the specified CID.

## Integration Object Objects

The integration object contains one or more integration components. The following integration object objects are provided:

- CSSEAIIntObjIn
- CSSEAIIntObjOut

### CSSEAIIntObjIn

This object represents an input integration object, open for reading, that is contained in the integration message. The integration object has a name and contains zero or more instances of actual integration objects. Integration object instances are accessed one at a time, similar to accessing database records. Each instance has a primary integration component that contains data and every subordinate integration components. The object provides the *Exists*, *FirstInstance*, *GetPrimaryIntComp*, and *NextInstance* methods.

#### Exists() Method

This method checks to see if the integration object is actually present in the input data. It takes no parameters.

**Syntax**     *Exists*()

**Returns**     Boolean

**Usage**     Call *Exists* after retrieving the integration object from the integration message. If the integration object was found and is open for reading, the *Exists* method returns `true`.

#### FirstInstance() Method

This method moves to the first integration object instance and sets it as the active instance.

**Syntax**     *FirstInstance*()

**Returns**     Boolean

**Usage**     The *FirstInstance* method returns `true` if the instance exists, `false` otherwise.



## GetPrimaryIntComp() Method

This method returns the primary integration component of the active instance of the integration object. [Table 19](#) lists the parameter for this method.

**Syntax**    `GetPrimaryIntComp(name)`

**Table 19. Parameter for GetPrimaryIntComp() Method**

| Parameter         | Description  |
|-------------------|--|
| <code>name</code> | The name of a primary integration component in the active integration object instance. |

**Returns**    *CSSEAIPrimaryIntCompIn* Input Primary Integration Component

**Usage**    Gets the primary integration component of the active instance of the integration object and opens it for input.

This method always returns an input primary integration component object, even if the component does not exist. Call the *Exists* method on the returned object to test for this condition. If there is no active instance, a call to this method raises an error.

## NextInstance() Method

This method moves a pointer to the next logical integration object instance in the active integration message.

**Syntax**    `NextInstance()`

**Returns**    Boolean

**Usage**    Moves to the next integration object instance and makes it the active instance. This method returns `true` if the instance exists, or `false` if there are no more instances. If neither the *NextInstance* or the *FirstInstance* method has been called previously, the *NextInstance* method moves to the first instance in the message.

# CSSEAllIntObjOut

This object represents an output integration object, open for writing, that is contained in the integration message. It provides `CreatePrimaryIntComp` and `NewInstance` methods as an interface to the output integration object.

## CreatePrimaryIntComp() Method

This method creates a new primary integration component. [Table 20](#) lists the parameter for this method.

**Syntax**      `CreatePrimaryIntComp(name)`

**Table 20.    Parameter for CreatePrimaryIntComp() Method**

| Parameter | Description  |
|-----------|--|
| name      | Assigned as the name of the Primary Integration Component. |

**Returns**      `CSSEAllPrimaryIntCompOut` Primary Integration Component, open for output

**Usage**      Use the *Exists* method to test for existence of the integration object instance, then create a new integration object instance and set it as the active instance, using the *NewInstance* method. You must perform these tasks before calling the `CreatePrimaryIntComp()` method.

## NewInstance() Method

This method creates a new instance of an integration object and makes it the active instance.

**Syntax**      `NewInstance()`

**Returns**      Not applicable

## Primary Integration Component Objects

A *primary integration component* represents the integration component contained within an integration object instance. It has a name and contains records with data from actual integration components. Each record may have fields and subordinate integration components.

The following primary integration component objects are provided:

- CSSEAIPrimaryIntCompIn
- CSSEAIPrimaryIntCompOut

### CSSEAIPrimaryIntCompIn

This object represents the input primary integration component, open for reading. Your data transformation maps can use this object's methods to traverse integration components. The object provides *Exists*, *FirstRecord*, *GetFieldValue*, *GetIntComp*, and *NextRecord* methods:

#### Exists() Method

This method checks to see if the primary integration component is actually present in the input data. It takes no parameters.

**Syntax**     `Exists()`

**Returns**     `Boolean`

**Usage**     Call *Exists* after retrieving the primary integration component with the `CSSEAIIntObjIn.GetPrimaryIntComp` method, and before invoking the primary integration component's other methods.

If the primary integration component was found and is open for reading, the *Exists* method returns `true`.

**FirstRecord() Method**

This method moves a pointer to the first component record in the primary integration component.

**Syntax**      FirstRecord()

**Returns**      Boolean

**Usage**      Moves to the first integration component record and sets it as the active record. This method returns `true` if the record exists, `false` if the integration component has no records.

**GetFieldValue() Method**

This method returns the value of the primary integration component field from the active record. [Table 21](#) lists the parameters for this method.

**Syntax**      GetFieldValue(*name* [, *defaultIfNull* [, *defaultIfEmpty*]])

**Table 21.    Parameters for GetFieldValue() Method**

| Parameter      | Description  |
|----------------|--|
| name           | The name of a primary integration component field.                       |
| defaultIfNull  | Optional. Sets the default value if the field does not exist.            |
| defaultIfEmpty | Optional. Sets the default value if the field is set to an empty string. |

**Returns**      String or `null`

**Usage**      A `null` value is returned if the active record does not contain the field. Otherwise, a string containing the value in the field is returned. If there is no active record, this method raises an error.

If the named argument does not exist, `null` is returned. If the named argument exists but the value is the empty string, the empty string is returned. You can use the *defaultIfNull* and *defaultIfEmpty* optional arguments to change this behavior.

The arguments *defaultIfNull* and *defaultIfEmpty* are optional; however, if you specify *defaultIfEmpty*, you must also specify the *defaultIfNull* argument.

## GetIntComp() Method

This method returns the named integration component from the active record and opens it for input. [Table 22](#) lists the parameter for this method.

**Syntax**     `GetIntComp(name)`

**Table 22. Parameter for GetIntComp() Method**

| Parameter         | Description  |
|-------------------|--|
| <code>name</code> | The name of an integration component in the active record. |

**Returns**     *CSSEAllIntCompIn* Input Integration Component

**Usage**     This method always returns an input integration component object, even if the component does not exist. Call the *Exists* method on the returned object to test for this condition. If there is no active record, a call to this method raises an error.

## NextRecord() Method

This method moves a pointer to the next logical record in the active integration component.

**Syntax**     `NextRecord()`

**Returns**     Boolean

**Usage**     Moves to the next record and makes it the active record. Returns `true` if the record exists, or `false` if there are no more records. Moves to the first record if neither the *NextRecord* method nor the *FirstRecord* method has been called previously.

# CSSEAIPrimaryIntCompOut

This object represents the output primary integration component. You can use the object's methods to create output integration components and records and to copy input data records to output data records. The object provides *CopyFieldValue*, *CreateIntComp*, *NewRecord*, *SetCopySource*, and *SetFieldValue* methods.

## CopyFieldValue() Method

This method sets the value of a field in the active record to the value of a field in the current source record. [Table 23](#) lists the parameters for this method.

**Syntax**     *CopyFieldValue(targetName, sourceName [, defaultIfNull [, defaultIfEmpty]])*

**Table 23.   Parameters for CopyFieldValue() Method**

| Parameter      | Description  |
|----------------|--|
| targetName     | Name of the field to set in the output integration component.  |
| sourceName     | Name of the field to retrieve from the input integration component.  |
| defaultIfNull  | Optional value that specifies what should be inserted into the target, if the source field does not exist. |
| defaultIfEmpty | Optional value that specifies what to use as a source value if the source field is empty.                  |

**Returns**     Not applicable

**Usage**     Use this method to copy a field from an input integration component to the output primary integration component. You could achieve the same results by calling the *GetFieldValue* method on the input component and the *SetFieldValue* on the output component; however, using *CopyFieldValue* is easier.

You must call the *SetCopySource* method first to specify the source integration component. *CopyFieldValue* uses the active records of the input and output components of the active integration component.

If the integration component is not set with the *SetCopySource* method first, a call to the *CopyFieldValue* method raises an error. An error also occurs if either input or output component does not have an active record.

If you set the copy source using the following statement:

```
outIntComp.SetCopySource (inIntComp);
```

the following two statements are equivalent:

```
outIntComp.SetFieldValue("Fld-A", inIntComp.GetFieldValue("X"));
```

```
outIntComp.CopyFieldValue("Fld-A", "X");
```

Using the second convention is convenient if you are copying many fields between the same components.

## CreateIntComp() Method

This method creates a new integration component. [Table 24](#) lists the parameters for this method.

**Syntax** CreateIntComp(*name* [, *createNow*])

**Table 24. Parameters for CreateIntComp() Method**

| Parameter | Description  |
|-----------|--|
| name      | The name of the new integration component.   |
| createNow | Optional. By default, the underlying data object is created in the output data object at the time this method is called. To change this behavior, specify the optional createNow argument as <i>false</i> . If you specify <i>createNow</i> as <i>false</i> , the underlying data object is not created until you make the first <i>NewRecord</i> call on the newly created integration component. Defaults to <i>true</i> . |

**Returns** CSSEAllIntCompOut. Output Integration Component

**Usage** Use this method to create a new integration component, open it for writing, and add it to the active record of the integration component.

**NOTE:** This method raises an error if you call it without an active integration component record. Use the *NewRecord* method to create a new record and set the active record.

**NewRecord() Method**

This method creates a new record in a primary integration component.

**Syntax**      NewRecord()

**Returns**      Not applicable

**Usage**      This method adds a new primary integration component record and makes it the active record.

**SetCopySource() Method**

This method establishes the integration component from which a field value will be copied. [Table 25](#) lists the parameter for this method.

**Syntax**      SetCopySource(*IntComp*)

**Table 25.    Parameter for SetCopySource() Method**

| Parameter | Description  |
|-----------|--|
| IntComp   | The integration component object—either CSSEAIPrimaryIntCompIn or CSSEAIIntCompIn. |

**Returns**      Not applicable

**Usage**      Call this method prior to a call to the *CopyFieldValue* method.



## SetFieldValue() Method

This method sets the value of the named field in the active integration component record. [Table 26](#) lists the parameters for this method.

**Syntax**     SetFieldValue(*name*, *value*)

**Table 26. Parameters for SetFieldValue() Method**

| Parameter    | Description  |
|--------------|--|
| <i>name</i>  | The name of a field in the active record of the primary integration component. |
| <i>value</i> | The string value to be put into the field given in the <i>name</i> parameter.  |

**Returns**     Not applicable

**Usage**       Both the name and value arguments should be strings.

The field is not set if the value is `null`. This method provides no return value.

This method raises an error if called while there is no active record.

---

**NOTE:** Siebel eScript automatically converts most types to strings as necessary.

---

## Integration Component Objects

An *integration component object* represents integration components. The following integration component objects are provided:

- CSSEAIIntCompIn
- CSSEAIIntCompOut

### CSSEAllIntCompln

This object represents the input integration component, open for reading. You can use the object's methods to traverse actual integration components and to retrieve data from those integration components. The object provides `Exists`, `FirstRecord`, `GetFieldValue`, `GetIntComp`, and `NextRecord` methods.

#### **Exists() Method**

This method checks to see if the integration component is actually present in the input data. It takes no parameters.

**Syntax**      `Exists()`

**Returns**      `Boolean`

**Usage**      Call *Exists* after retrieving the integration component from its parent object using the `GetIntComp` method, and before invoking the integration component's other methods.

If the integration component is found and is open for reading, the *Exists* method returns `true`.

#### **FirstRecord() Method**

This method moves a pointer to the first component record in the integration component.

**Syntax**      `FirstRecord()`

**Returns**      `Boolean`

**Usage**      Moves to the first integration component record and sets it as the active record. This method returns `true` if the record exists, `false` if the integration component has no records.

## GetFieldValue() Method

This method returns the value of the integration component field from the active record. [Table 27](#) lists the parameters for this method.

**Syntax**     `GetFieldValue(name [, defaultIfNull [, defaultIfEmpty]])`

**Table 27. Parameters for GetFieldValue() Method**

| Parameter                   | Description   |
|-----------------------------|---|
| <code>name</code>           | The name of an integration component field.                       |
| <code>defaultIfNull</code>  | Optional. Value to return if the field does not exist.            |
| <code>defaultIfEmpty</code> | Optional. Value to return if the field is set to an empty string. |

**Returns**     String or `null`

**Usage**     A `null` value is returned if the active record does not contain the field. Otherwise, a string containing the value in the field is returned. If there is no active record, this method raises an error.

If the named argument does not exist, `null` is returned. If the named argument exists but the value is the empty string, the empty string is returned. You can use the *defaultIfNull* and *defaultIfEmpty* arguments to change this behavior.

---

**NOTE:** The arguments *defaultIfNull* and *defaultIfEmpty* are optional. However, if you specify *defaultIfEmpty*, you must also specify the *defaultIfNull* argument.

---

**GetIntComp() Method**

This method returns the integration component from the active record and opens it for input. [Table 28](#) lists the parameter for this method.

**Syntax**     `GetIntComp(name)`

**Table 28.    Parameter for GetIntComp() Method**

| Parameter         | Description  |
|-------------------|--|
| <code>name</code> | The name of an integration component in the active record. |

**Returns**     *CSSEAllIntCompIn* Input Integration Component

**Usage**        This method always returns an input integration component object, even if the component does not exist. Call the *Exists* method on the returned object to test for this condition.

---

**NOTE:** If there is no active record, a call to this method raises an error.

---

**NextRecord() Method**

This method moves a pointer to the next logical record in the active integration component.

**Syntax**     `NextRecord()`

**Returns**     `Boolean`

**Usage**        Moves to the next record and makes it the active record. Returns `true` if the record exists, or `false` if there are no more records. Moves to the first record if neither the *NextRecord* method nor the *FirstRecord* method has been called previously.

## CSSEAllIntCompOut

This object represents the output integration object, open for writing. You can use this object's methods to create new output integration components and to copy or set actual data in the records of the integration components. The object provides *CopyFieldValue*, *CreateIntComp*, *NewRecord*, *SetCopySource*, and *SetFieldValue* methods.

### CopyFieldValue() Method

This method sets the value of a field in the active record to the value of a field in the current source record. [Table 29](#) lists the parameters for this method.

**Syntax** `CopyFieldValue(targetName, sourceName [, defaultIfNull [, defaultIfEmpty]])`

**Table 29. Parameters for CopyFieldValue() Method**

| Parameter      | Description  |
|----------------|--|
| targetName     | Name of the field to set in the output integration component.  |
| sourceName     | Name of the field to retrieve from the input integration component.  |
| defaultIfNull  | Optional value that specifies what should be inserted into the target, if the source field does not exist. |
| defaultIfEmpty | Optional value that specifies what to use as a source value if the source field is empty.                  |

**Returns** Not applicable

**Usage** Use this method to copy a field from an input integration component to the output integration component. You could achieve the same results by calling the *GetFieldValue* method on the input component and the *SetFieldValue* on the output component; however, using *CopyFieldValue* is easier.

You must call the *SetCopySource* method first to specify the source integration component. *CopyFieldValue* uses the active records of the input and output components of the active integration component.

If the integration component is not set with the *SetCopySource* method first, a call to the *CopyFieldValue* method raises an error. An error also occurs if either input or output component does not have an active record.

If you set the copy source using the following statement:

```
outIntComp.SetCopySource (inIntComp);
```

the following two statements are equivalent:

```
outIntComp.SetFieldValue("Fld-A", inIntComp.GetFieldValue("X"));
outIntComp.CopyFieldValue("Fld-A", "X");
```

Using the second convention is convenient if you are copying many fields between the same components.

## CreateIntComp() Method

This method creates a new integration component. [Table 30](#) lists the parameters for this method.

**Syntax**    `CreateIntComp(name [, createNow])`

**Table 30. Parameters for CreateIntComp() Method**

| Parameter | Description   |
|-----------|---|
| name      | The name of the new integration component.  |
| createNow | Optional. By default, the underlying data object is created in the output data object at the time this method is called. To change this behavior, specify the optional <i>createNow</i> argument as <i>false</i> . If you specify <i>createNow</i> as <i>false</i> , the underlying data object is not created until you make the first <i>NewRecord</i> call on the newly created integration component. Defaults to <i>true</i> . |

**Returns**    *CSSEAllIntCompOut*. Output Integration Component

**Usage**    Use this method to create a new integration component, open it for writing, and add it to the active record of the integration component.

This method raises an error if you call it without an active integration component record. Use the *NewRecord* method to create a new record and set the active record.

## SetCopySource() Method

This method establishes the integration component from which a field value will be copied. [Table 31](#) lists the parameter for this method.

**Syntax**    `SetCopySource(IntComp)`

**Table 31. Parameter for SetCopySource() Method**

| Parameter | Description  |
|-----------|--|
| IntComp   | The integration component object—either CSSEAIPrimaryIntCompIn or CSSEAIIntCompIn. |

**Returns**    Not applicable

**Usage**    Call this method prior to a call to the *CopyFieldValue* method.

## SetFieldValue() Method

This method sets the value of the named field in the active integration component record. [Table 32](#) lists the parameters for this method.

**Syntax**    `SetFieldValue(name, value)`

**Table 32. Parameters for SetFieldValue() Method**

| Parameter | Description   |
|-----------|---|
| name      | The name of a field in the active record of the integration component.        |
| value     | The string value to be put into the field given in the <i>name</i> parameter. |

**Returns**    Not applicable

**Usage**    Both the name and value arguments should be strings.

The field is not set if the value is `null`. This method provides no return value.

This method raises an error if called while there is no active record.

---

**NOTE:** Siebel eScript automatically converts most types to strings as necessary.

---

## MIME Message Objects and Methods

Siebel eAI represents MIME documents using a property set format. This is the format used by the EAI MIME Doc Converter Business Service. The objects and methods described here provide access to this property set format, and are intended for use in conjunction with transforming pieces of the MIME message to and from Siebel Integration Messages.

---

**NOTE:** The EAI MIME Hierarchy Converter Business Service is the preferred method of converting between the property set representation Siebel Messages.

---

The following MIME message objects are provided:

- CSSEAIMimeMsgIn
- CSSEAIMimeMsgOut

### CSSEAIMimeMsgIn

This object represents an input MIME Message, open for reading. The MIME message is in the property set format generated by the EAI MIME Doc Converter. The object consists of a series of MIME parts forming the different pieces of the message.

This object provides GetArgument, GetPartCount, GetPart, GetPartByCID, GetAttachmentByCID, and GetXMLRootPart methods:



## GetArgument ()Method

This method gets the value of a business service argument. For example, this could get the name of a map function in the business service. [Table 33](#) lists the parameters for this method.

**Syntax**     `GetArgument(name [, defaultIfNull [, defaultIfEmpty]])`

**Table 33. Parameters for GetArgument() Method**

| Parameter      | Description  |
|----------------|--|
| name           | The name of a business service argument.                             |
| defaultIfNull  | Returned if a service argument of the specified name does not exist. |
| defaultIfEmpty | Returned if the service argument is set to an empty string.          |

**Returns**     String or null

**Usage**     Use this method to get the value of an argument passed to the business service. For example, if the `MapName` argument passed to the business service is *MapExtOrderToOrder*, the call:

```
intMsgIn.GetArgument ( "MapName" );
```

returns the name of the map, *MapExtOrderToOrder*, passed to the business service.

If the named argument does not exist, `null` is returned. If the named argument exists but the value is the empty string, the empty string is returned. You can use the *defaultIfNull* and *defaultIfEmpty* optional arguments to change this behavior.

The arguments *defaultIfNull* and *defaultIfEmpty* are optional; however, if you specify *defaultIfEmpty*, you must also specify the *defaultIfNull* argument.

**GetPartCount () Method**

This method returns the number of parts in the MIME message. [Table 34](#) lists the parameter for this method.

**Syntax**      GetPartCount()

**Returns**      This method returns the number of parts in the MIME message.

**GetPart () Method**

**Syntax**      GetPart(*index*)

**Table 34.    Parameters for GetPart() Method**

| Parameter | Description                       |
|-----------|-----------------------------------|
| index     | Index of the MIME part to return. |

**Returns**      Property set. Returns the part, a property set, specified by the index. The index is zero based. Returns null if the index is out of bounds.

**GetPartByCID () Method**

Retrieve a MIME part based on the MIME Content Identifier (CID). [Table 35](#) lists the parameter for this method.

**Syntax**      GetPartByCID(*cid*)

**Table 35.    Parameters for GetPartByCID() Method**

| Parameter | Description                          |
|-----------|--------------------------------------|
| cid       | MIME Content Identifier to retrieve. |

**Returns**      Returns null if there is no part with the specified CID.

## GetAttachmentByCID () Method

The same functionality as *CSSEAIMimeMsgIn.GetPartByCID*. Supports using a *CSSEAIMimeMsgIn* as an attachment source for copying attachments to output objects. [Table 36](#) lists the parameter for this method.

**Syntax**     `GetAttachmentByCID(cid)`

**Table 36. Parameters for GetAttachmentByCID() Method**

| Parameter        | Description                          |
|------------------|--------------------------------------|
| <code>cid</code> | MIME Content Identifier to retrieve. |

**Returns**     The attachment (a property set) specified by the CID. Returns null if there is no attachment with the specified CID.

## GetXMLRootPart () Method

Finds the first MIME part that is an XML message in property set format and returns the root element of the XML document. The XML message must be in property set format as produced by the XML Hierarchy Converter Business Service. An error is raised if the XML message is not found. The method is intended for use with MIME messages that consist of an XML message and a series of related attachments. The property set returned is consistent with what *XPSGetRootElement* returns, and can be accessed with the XML Property Set functions. See [“XML Property Set Functions” on page 129](#).

**Syntax**     `GetXMLRootPart()`

**Returns**     MIME body part representing an XML document.

## CSSEAIMimeMsgOut

This object represents an output MIME message, open for writing. The object provides `SetArgument`, `CreateXMLPart`, `SetAttachmentSource`, and `CopyAttachment` methods:

### SetArgument() Method

This method sets the value of a business service argument. [Table 37](#) lists the parameters for this method.

**Syntax**     `SetArgument(name, value)`

**Table 37. Parameters for SetArgument() Method**

| Parameter | Description  |
|-----------|--|
| name      | The name of an argument in the active business service.                            |
| value     | The string value corresponding to the argument named by the <i>name</i> parameter. |

**Returns**     Not applicable

**Usage**     You can call the *SetArgument* method to establish the value of a given output argument for the business service method invocation.

## CreateXMLPart () Method

This method is similar to `XPSCreateRootElement`. See [“XPSCreateRootElement\(\)” on page 130](#). It creates an XML MIME part and adds it to the MIME document. The property set representing the XML root element is returned. The property set returned can be populated using the XML Property Set functions. See [“XML Property Set Functions” on page 129](#). [Table 38](#) lists the parameter for this method.

**Syntax** `CreateXMLPart(xmlRootTagName)`

**Table 38. Parameters for CreateXMLPart() Method**

| Parameter      | Description   |
|----------------|---|
| xmlRootTagName | The name you want to supply as the root element name in the XML document. |

**Returns** Property set

## SetAttachmentSource () Method

This method establishes the source object from which to copy attachment objects. The source object must be a `CSSEAIIntMsgIn`, `CSSEAIMimeMsgIn`, or other object implementing the `GetAttachmentByCID` method. [Table 39](#) lists the parameter for this method.

**Syntax** `SetAttachmentSource(source)`

**Table 39. Parameters for SetAttachmentSource() Method**

| Parameter | Description            |
|-----------|------------------------|
| source    | The attachment source. |

**CopyAttachment () Method**

This method copies an attachment from the attachment source to the output MIME message object. The attachment is referenced by the MIME Content Identifier (CID). The attachment copy, a property set, is returned. The attachment source must be established by calling *CSSEAIMimeMsgOut.SetAttachmentSource* prior to calling this method. [Table 40](#) lists the parameter for this method.

**Syntax**      CopyAttachment(*cid*)

**Table 40.    Parameters for CopyAttachment() Method**

| Parameter | Description  |
|-----------|--|
| cid       | MIME Content Identifier of the attachment to copy. |

**Returns**      Property set. This method returns null if the attachment source does not contain an attachment with the specified CID.

## Attachments and Content Identifiers in MIME Messages

A MIME message contains one or more parts, each representing a separate piece of the message. One common use of multipart MIME messages is to include attachments with a message.

---

**NOTE:** All the examples have to be typed single-spaced and without word wrap.

---

Each MIME body part has an optional Content Identifier (CID) used to identify it. The Content Id is part of the MIME part header, for example:

```
--unique_boundary_123

Content-Type : image/jpeg

Content-ID : <001110.102215@abc.com>
```

Then the CID is 001110.102215@abc.com. The CID is usually referenced from another part of the MIME message. A common scheme is to use an XML document as the main part of the MIME message, and use Content IDs to reference the other attachments in the message. The following is an example of a MIME message with attachment.

```
MIME-Version: 1.0

Content-Type: multipart/related;
    boundary="unique_boundary_123";
    type="application/xml"

Content-Transfer-Encoding: binary

--unique_boundary_123

Content-Type: application/xml; charset="UTF-8"

Content-Transfer-Encoding: binary

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

<!DOCTYPE Memo SYSTEM "Memo.dtd">

<Memo>
```

```
<To>All Employees</To>

<Subject>Map and Directions</Subject>

<Body>Maps to company headquarters are attached.</Body>

<ListOfAttachments>

  <Attachment>

    <URI>cid:001110.102203@siebel.com</URI>

    <Filename>largemap.jpg</Name>

  </Attachment>

  <Attachment>

    <URI>cid:001110.102211@siebel.com</URI>

    <Filename>detailmap.jpeg</Filename>

  </Attachment>

</ListOfAttachment>

</Memo>

--unique_boundary_123
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <001110.102203@siebel.com>

  [... Raw JPEG Image ...]

--unique_boundary_123
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <001110.102211@siebel.com>

  [... Raw JPEG Image ...]

--unique_boundary_123-
```



# XML Property Set Functions

Siebel eAI represents XML documents using the property set format. While Siebel eAI does not always require using the property set format, this representation is used by EAI Business Services such as the EAI XML Converter. The functions described in this section provide a simple interface for manipulating XML documents using the property set format.

## Top-Level Property Set Functions

These functions are used to manipulate the top-level property set passed to the Map function.

### XPSGetRootElement()

This function returns the property set representing the root element of the XML document. If the root element is not present, the system raises an error. [Table 41](#) lists the parameter for this function.

**Syntax** XPSGetRootElement(*xmlPropSetIn*)

**Table 41. Parameter for XPSGetRootElement() Method**

| Parameter    | Description   |
|--------------|---|
| xmlPropSetIn | The name of the property set representing the root element of the XML document. |

**Returns** Property set

**Usage** Use this function to return the root element of an XML document.

**XPSCreateRootElement()**

This function creates the root element in an output XML document and returns the property set representing it. The element tag in the XML document is set to the value of the *tagName* argument. [Table 42](#) lists the parameters for this function.

**Syntax**      `XPSCreateRootElement(xmlPropSetOut, tagName)`

**Table 42. Parameters for XPSCreateRootElement() Method**

| Parameter     | Description   |
|---------------|---|
| xmlPropSetOut | The output property set.  |
| tagName       | The name you want to supply as the root element name in the XML document. |

**Returns**      Property set

**Usage**          Use this function to create the root element of an XML document that represents a property set. Because the root element does not directly map to a component in the property set, you can give it any representative name.

As an example of how the two prior functions work, consider the following XML document:

```
<?xml version="1.0"?>
<!DOCTYPE LETTER SYSTEM "letter.dtd">
<letter>
  <from>Mary Smith</from>
  <to>Paul Jones</to>
  <text>Hello!</text>
</letter>
```

The root element is `<letter>`. The property set for the `<letter>` element can be retrieved from the input property set using *EAIXPS\_GetRootElement*, or it can be created in the output property set using *EAIXPS\_CreateRootElement*.

A map function that converts a letter to a memo might start with the following code:

```
function ConvertLetterToMemo (xmlPropSetIn, xmlPropSetOut)
{
  var xmlLetter = XPSCreateRootElement (xmlPropSetIn);
```

```

var xmlMemo = XPSCreateRootElement (xmlPropSetOut, "memo");

... Code to fill in the 'memo' from the 'letter' ...
}

```

## XML Element Accessors

These functions provide access to elements represented by property sets. [Table 43](#) lists the parameter for this function.

### XPSTagName()

Retrieves the tag name of an XML element.

**Syntax** `XPSTagName (xmlPropSet)`

**Table 43. Parameter for XPSTagName() Method**

| Parameter  | Description              |
|------------|--------------------------|
| xmlPropSet | The output property set. |

**Returns** String. If *xmlPropSet* is null, *XPSTagName* returns null.

### XPSSetTagName()

This function sets the tag name of an XML element. [Table 44](#) lists the parameters for this function.

**Syntax** `XPSSetTagName (xmlPropSet, tagName)`

**Table 44. Parameters for XPSSetTagName() Method**

| Parameter  | Description  |
|------------|--|
| xmlPropSet | The property set.  |
| tagName    | The name you want to supply as the current element name in the XML document. |

**Returns** String

**XPSGetTextValue()**

This function returns the text value of an XML element as a string. [Table 45](#) lists the parameters for this function.

**Syntax**     XPSGetTextValue (*xmlPropSet* [, *defaultIfNull* [, *defaultIfEmpty*]] )

**Table 45.   Parameters for XPSGetTextValue() Method**

| Parameter      | Description   |
|----------------|---|
| xmlPropSet     | The output property set.  |
| defaultIfNull  | Specify a value to override the null default value that results if <i>xmlPropSet</i> is null. |
| defaultIfEmpty | Specify a value to override an empty string ("") contained in <i>xmlPropSet</i> .             |

**Returns**     String or null

**Usage**       If *xmlPropSet* is null then null is returned. You can use the optional *defaultIfNull* and *defaultIfEmpty* arguments to override null and empty string ("") return values. An element’s text value is the text between an XML element's start and end tags, excluding child elements.

**XPSSetTextValue()**

This function sets the text value of an XML element. [Table 46](#) lists the parameters for this function.

**Syntax**     XPSSetTextValue (*xmlPropSet*, *text*)

**Table 46.   Parameters for XPSSetTextValue() Method**

| Parameter  | Description  |
|------------|--|
| xmlPropSet | The property set.  |
| text       | A string you want inserted between start and end tags of an XML element. |

**Returns**     Not applicable

**Usage** The text argument should be a string. An element's text value is the text between the element's start and end tags, excluding child elements.

### **XPSGetAttribute()**

This function retrieves an element's attribute of the given name and returns it as a string. [Table 47](#) lists the parameters for this function.

**Syntax** `XPSGetAttribute (xmlPropSet, name [, defaultIfNull [, defaultIfEmpty]])`

**Table 47. Parameters for XPSGetAttribute() Method**

| Parameter      | Description   |
|----------------|---|
| xmlPropSet     | The output property set.  |
| name           | The name you want to supply as the root element name in the XML document.                     |
| defaultIfNull  | Specify a value to override the null default value that results if <i>xmlPropSet</i> is null. |
| defaultIfEmpty | Specify a value to override an empty string ("") contained in <i>xmlPropSet</i> .             |

**Returns** String

**Usage** A `null` value is returned if *xmlPropSet* is null or the element does not have the named attribute. The optional *defaultIfNull* and *defaultIfEmpty* arguments can be used to override null and empty string ("") return values.

**XPSSetAttribute()**

This function sets an element attribute value. [Table 48](#) lists the parameters for this function.

**Syntax**      `XPSSetAttribute (xmlPropSet, name, value)`

**Table 48.    Parameters for XPSSetAttribute() Method**

| Parameter  | Description   |
|------------|---|
| xmlPropSet | The output property set.                                |
| name       | Attribute name.   |
| value      | String value you want to supply as the attribute value. |

**Returns**      String

**Usage**        No action is taken if any of the arguments are null.

**XPSSetChildCount()**

This function returns the number of children of an element. [Table 49](#) lists the parameter for this function.

**Syntax**      `XPSSetChildCount(xmlPropSet)`

**Table 49.    Parameter for XPSSetChildCount() Method**

| Parameter  | Description       |
|------------|-------------------|
| xmlPropSet | The property set. |

**Returns**      Number

**Usage**        All children of an element are also elements.

XPSGetChild()

This function returns the *n*th child element as specified by the index. [Table 50](#) lists the parameters for this function.

**Syntax** XPSGetChild(*xmlPropSet*, *index*)

**Table 50. Parameters for XPSGetChild() Method**

| Parameter  | Description  |
|------------|--|
| xmlPropSet | The property set.  |
| index      | Number, starting at zero, of child elements of another element in an XML document. |

**Returns** Property set

**Usage** Child elements are specified using a zero-based index. A value of `null` is returned if the index is invalid.

XPSFindChild()

This function returns the first child element with the *tagName*. [Table 51](#) lists the parameters for this function.

**Syntax** XPSFindChild (*xmlPropSet*, *tagName*)

**Table 51. Parameters for XPSFindChild() Method**

| Parameter  | Description   |
|------------|---|
| xmlPropSet | The property set.   |
| tagName    | An XML element tag that signifies the first child element of another XML element. |

**Returns** Property set.

**Usage** A value of `null` is returned if there is no child with the specified tag name.

XPSAddChild()

This function creates a new child element with the *tagName* and appends it to the list of *xmlPropSet*'s children. [Table 52](#) lists the parameters for this function.

**Syntax** XPSAddChild (*xmlPropSet*, *tagName* [, *textValue*])

**Table 52. Parameters for XPSAddChild() Method**

| Parameter  | Description   |
|------------|---|
| xmlPropSet | The property set.                                   |
| tagName    | The name you want to give to the new child element. |
| textValue  | Optional. Sets the text value of the new element.   |

**Returns** Property set



## Examples

The following example converts a `<letter>` to a `<memo>`.

---

**NOTE:** The input letter in this example is slightly different from the previous example.

---

The input XML document is:

```
<letter
  from="Mary Smith"
  to="Paul Jones">
  <text>Hello!</text>
</letter>
```

The conversion function converts this to a memo format, as follows:

```
<memo>
  <type>Interoffice Memo</type>
  <header>
    <from>Mary Smith</from>
    <to>Paul Jones</to>
  </header>
  <body>Hello!</body>
</memo>
```

The map function that performs this conversion is shown below:

```
function ConvertLetterToMemo (xmlPropSetIn, xmlPropSetOut)
{
  var letter = XPSGetRootElement (xmlPropSetIn);
  var memo = XPSCreateRootElement (xmlPropSetOut, "memo");
  XPSAddChild (memo, "type", "Interoffice Memo");
  var header = XPSAddChild (memo, "header");
  XPSAddChild (header, "from", XPSGetAttribute (letter,
"from"));
  XPSAddChild (header, "to", XPSGetAttribute (letter, "to"));
  XPSAddChild (memo, "body", XPSGetTextValue (XPSFindChild
(letter, "text")));
}
```

# EAI Value Maps

EAI Value Maps correlate Siebel data values with external data values.

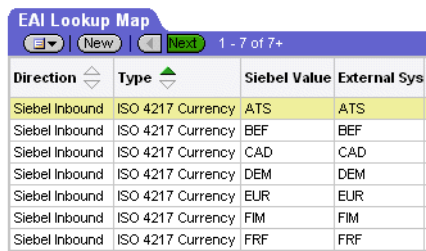
If you are:

- Sending and receiving data, you can create inbound and outbound maps for the same data
- Receiving data only, you need only to define an inbound map
- Sending data only, you need only to define an outbound map

Consider an example of how EAI Value Maps provide correlations between Siebel applications and the SAP R/3 system. SAP country codes, which are represented as two-character codes, are different from Siebel country codes, represented by the country name spelled out. An EAI Value Map provides a lookup table that lists these two sets of data side by side.

The EAI Value Map entries are stored in the EAI Value Map table. You can view and administer this table from the EAI Value Maps view in the Integration Administration screens in the Siebel client. The Siebel client groups the entries logically based on the Type and Direction columns.

Figure 11 shows the entries form two logical groupings, with entries for the Siebel inbound and Siebel outbound entries.



The screenshot shows a window titled "EAI Lookup Map" with a toolbar containing "New" and "Next" buttons. Below the toolbar is a table with four columns: "Direction", "Type", "Siebel Value", and "External Sys". The table contains seven rows of data, all with "Siebel Inbound" in the "Direction" column and "ISO 4217 Currency" in the "Type" column. The "Siebel Value" and "External Sys" columns contain two-letter country codes: ATS, BEF, CAD, DEM, EUR, FIM, and FRF.

| Direction      | Type              | Siebel Value | External Sys |
|----------------|-------------------|--------------|--------------|
| Siebel Inbound | ISO 4217 Currency | ATS          | ATS          |
| Siebel Inbound | ISO 4217 Currency | BEF          | BEF          |
| Siebel Inbound | ISO 4217 Currency | CAD          | CAD          |
| Siebel Inbound | ISO 4217 Currency | DEM          | DEM          |
| Siebel Inbound | ISO 4217 Currency | EUR          | EUR          |
| Siebel Inbound | ISO 4217 Currency | FIM          | FIM          |
| Siebel Inbound | ISO 4217 Currency | FRF          | FRF          |

**Figure 11. EAI Value Maps for Country Codes**

The Direction field determines the direction of the mapping and is either Siebel Outbound or Siebel Inbound. In a Siebel Outbound mapping, the Siebel Value field is the lookup key; the External System Value is the translation. In a Siebel Inbound mapping, the External System Value field is the lookup key; the Siebel Value is the translation.

You can add, remove, or modify entries in the Type group on the EAI Lookup Map view in the Siebel client. The EAI\_LOOKUP\_MAP\_TYPE list of values defines type values. You can modify the list from the Application Administration views in the Siebel client.

---

**NOTE:** You cannot change the values of the Direction field, which must be Siebel Outbound or Siebel Inbound.

---

The data transformation methods include an interface to EAI Value Maps for translating the codes of one database to another. You use the `EAIGetValueMap` function to obtain an interface to the mappings of specific Type-Direction pairs. You use the interface object's `Translate` method to find specific keys in the Type-Direction map and retrieve the translated values.

## EAIGetValueMap Function

You use the following statement in your Siebel eScript code to return a value map:

```
EAIGetValueMap (type, direction [,unmappedKeyHandler])
```

This object returns a value map for translating lookup keys using the Type-Direction combination.

- The type argument is a string found in the Type field of the EAI Value Map table.
- The direction argument must be either Siebel Inbound or Siebel Outbound string values.

A call to this function returns a *CSSEAIValueMap* object.

You can use the optional *unmappedKeyHandler* argument to control the behavior of the Translate method when it gets keys that do not have mappings in the table. The *unmappedKeyHandler* argument can be either a literal value or a function. If you pass a literal value, it is used as the default value. Otherwise, if you pass a function, the method calls that function, then uses the value returned by the function.

The *unmappedKeyHandler* defaults to an empty string ("").

## CSSEAIValueMap Translate Method

The *CSSEAIValueMap* object has one method: *Translate*. The Translate method takes one argument, as follows:

```
Translate (key)
```

The Translate method looks up the key value in the EAI Value Map and returns the translated value. The *EAIGetValueMap* call establishes the set of mappings for the translation using the type and direction arguments. The call looks for the key in either the Siebel Value column or in the External System Value column, depending on the value of the type argument.

- If the type is “Siebel Outbound,” the method returns the key found in the Siebel Value column. The translated value is in the External System Value column.
- If the type is “Siebel Inbound,” the method returns the key found in the External System Value column. The translated value is in the Siebel Value column.
- If *key* is *null* then the return value is *null*.
- If *key* is an empty string, the lookup is performed.

If there is no mapping, an empty string is returned.

If a nonempty string does not have a mapping, the *unmappedKeyHandler* value specified in the call to the *EAIGetValueMap* function is used to determine the translation.

## EAIGetValueMap unmappedKeyHandler Argument

The *unmappedKeyHandler* provides a flexible mechanism for handling cases where keys are not found in the EAI Value Map. In most situations, you can use literal values for defaults or you can use one of several predefined handler functions. However, you can also provide your own handler function.

The technique you use for handling unmapped values depends on the data being mapped.

Typical strategies include:

- Use the empty string as the translation.

This is the default strategy. This clears the field if the data is being imported into your Siebel application. To follow this approach, omit the *unmappedKeyHandler* argument or pass it as an empty string. For example:

```
var langMap = EAIGetValueMap("SAP Language","Siebel  
Inbound","");
```

This example looks up a nonexistent language code. This returns an empty string. For example:

```
langMap.Translate("ABC");// returns an empty string
```

- Use `null` as the translation.

This technique makes the result unspecified rather than empty. For data imported to Siebel applications, this keeps the existing value from being overridden when performing updates. Use `null` as the *unmappedKeyHandler*. For example:

```
var langMap = EAIGetValueMap("SAP Language","Siebel Inbound", null);
```

- Use a literal string as the translation.

Specify the string as the *unmappedKeyHandler*. For example:

```
var langMap = EAIGetValueMap("SAP Language","Siebel Inbound",  
"Unknown Language");
```

- Raise an error.

This may be the best strategy if the Value Map should contain mappings for every key. You can use the *EAIValueMap\_NoEntry\_RaiseError* function. For example:

```
var langMap = EAIGetValueMap ("SAP Language", "Siebel Inbound",  
EAIValueMap_NoEntry_RaiseError);
```

- Use the untranslated value.

The predefined function *EAIValueMap\_NoEntry\_ReturnLookupKey* implements this strategy. For example:

```
var langMap = EAIGetValueMap ("SAP Language", "Siebel Inbound",  
EAIValueMap_NoEntry_ReturnLookupKey);
```

Trying to look up a nonexistent language code (for example, “ABC”) will return the original key. For example:

```
langMap.Translate ("ABC"); // returns “ABC”
```

You can also write a custom handler function. You need to write a function taking three arguments: key, type, and direction. The value your function returns is used as the translation. For example:

```
function MyUnmappedLangHandler (key, type, direction)  
{  
    return ("Unknown Language: " + key);  
}  
  
var langMap = EAIGetValueMap ("SAP Language", "Siebel Inbound",  
MyUnmappedLangHandler);  
  
// Lookup a nonexistent language code.  
  
langMap.Translate ("ABC"); // returns "Unknown Language: ABC"
```

EAIGetValueMap() Method

This method retrieves objects for the required Type-Direction mapping. [Table 53](#) lists the parameters for this method.

**Syntax**    EAIGetValueMap(*type*, *direction* [, *unmappedKeyHandler*])

**Table 53.    Parameters for EAIGetValueMap() Method**

| Parameter          | Description  |
|--------------------|--|
| type               | Specifies the type of transformation map.  |
| direction          | A string specifying the direction of the message. The possible values are:<br>“Siebel Inbound”<br>“Siebel Outbound”                        |
| unmappedKeyHandler | Specifies the value to pass to the map for an unmapped key. Can be an empty string, null, a literal, or the name of a predefined function. |

**Returns**    An object you can use to access the EAI Value Maps.

**Usage**    Use this method at the beginning of a script function to retrieve objects for the required Type-Direction mapping. Then call the object’s Translate method to get the translation of a code from the map table as needed within the function.

**NOTE:** Providing a Type-Direction pair that does not have an entry in the EAI Value Map raises an error at the first call to the Translate method.

## Exception Handling Considerations

There are three categories of errors you might encounter in the data transformation area of your integration. These categories are:

- **Siebel errors.** Errors signaled by the built-in facilities that execute a map; for example, run-time Siebel eScript errors, business service invocation errors, BusComp errors, and errors in the data transformation functions.
  - Siebel errors are fatal, terminating execution of the map immediately.
  - The business service returns an error code other than OK. No specific error code is guaranteed, and they are not intended for workflow branching. Workflow processes can branch on the indication of an error occurrence, but not on a specific code.
  - The CSSService error stack will contain useful error information. In particular, data transformation function errors will generate error stacks describing the particular error.
- **User errors.** Errors signaled in custom maps using the *EAIRaiseErrorCode* call. These are similar to Siebel Framework errors, except that the map developer selects the error code and uses them for Workflow branching.
  - User errors are fatal, terminating execution of the map immediately.
  - The service returns the error code specified in the call to *EAIRaiseErrorCode*. Your workflow can branch on this code.
  - Available error codes are those in the Workflow generic error set.
  - You specify the entire error text for these generic errors in the call to *EAIRaiseErrorCode*.
  - You can use the function *EAIRaiseError* to raise an error without specifying a particular error code.
- **Map status flags.** The map developer can use the *SetArgument* method to set custom status information in the output property set. For example, you can use the *SetArgument* method to indicate that a required field is missing. This can be used for Workflow branching, if desired. This mechanism is independent of calls made to *EAIRaiseError*.



## Error Codes and Error Symbols

All errors each have an *error code*, which is a unique integer. A subset of errors also each have an error symbol. An *error symbol* is a text string that allows you to reference specific error codes in Siebel Workflow and in Siebel eScript. Errors that do not have an error symbol cannot be used for branch decisions and cannot be raised as user errors.

Error codes returned by a data transformation service may or may not have an associated error symbol. User errors will have error symbols. Currently, errors generated by Data Transformation Functions have error symbols. Errors occurring outside the data transformation framework often will not have error symbols.

## Data Transformation Error Processing

This section describes how the Data Transformation Functions handle errors, and how the top-level error code returned by the data transformation business service invocation is determined.

- **Framework errors occurring outside the Data Transformation Context.** These errors are passed through without change to the *CSSService* script invocation mechanism. That mechanism takes control and returns an error of its choice. For example, if your map invokes a BusComp and the BusComp signals an error, an exception is thrown that will be ignored by the Data Mapping Engine but passed to the *CSSService* script invocation mechanism, which sets up the error state and returns an error from the business service invocation.
- **Framework errors generated by Data Transformation Functions.** These are caught by an exception handler that sets up the state in the output PropertySet and passes control to the *CSSEAITEScriptService* class. *CSSEAITEScriptService* sets the error code on the business service as in the state, transforming error symbols to error codes in the process. Error symbols are specific to the failure.
- **User errors.** These are processed the same way as errors generated by the Data Transformation Functions, except that you specify the error symbols and error text in your maps.

# Exception Handling Functions

When writing your data transformation scripts, you can use the following functions to handle error conditions:

- EAIRaiseError
- EAIRaiseErrorCode
- EAIFormatMessage

**NOTE:** Before proceeding, read [“Exception Handling Considerations” on page 144](#).

## EAIRaiseError() Method

This method raises a fatal error and terminates the script. [Table 54](#) lists the parameters for this method.

**Syntax**     EAIRaiseError(*msg* [, *formatParameters*])

**Table 54.    Parameters for EAIRaiseError() Method**

| Parameter        | Description   |
|------------------|---|
| msg              | Error message text from the Data Mapping Engine.  |
| formatParameters | Optional string arguments inserted in the return value in the positions specified by the positional arguments in the <i>msg</i> parameter. A maximum of nine format parameters are allowed. |

**Usage**     You can provide format parameters to format the message text. For details, see [“EAIFormatMessage\(\) Method” on page 147](#).

## EAIRaiseErrorCode() Method

This method raises a fatal error, terminates the script, and returns an error symbol that it receives from the business service.

**Syntax** `EAIRaiseErrorCode(errorSymbol, msg)`

**Usage** You can use this function when you want to pass an error symbol to a workflow as an indication to branch on an exception. If you are not branching on the specific error code in your workflow, use *EAIRaiseError* instead.

## EAIFormatMessage() Method

This method formats strings that have position-independent arguments. [Table 55](#) lists the parameters for this method.

**Syntax** `EAIFormatMessage(msg [, formatParameters])`

**Table 55. Parameters for EAIFormatMessage() Method**

| Parameter                     | Description   |
|-------------------------------|---|
| <code>msg</code>              | A string that contains positional arguments. The substitution operation replaces the percent sign followed by a digit with the corresponding format parameter.                                    |
| <code>formatParameters</code> | Optional string arguments inserted in the return value in the positions specified by the positional arguments in the <code>msg</code> parameter. A maximum of nine format parameters are allowed. |

**Returns** A string of the *formatParameters* argument values in the positions specified by the positional arguments included in the *msg* parameter.

**Usage** You can use this function to generate messages from strings that are translated and whose positions have changed as a result of the translation.

**Example** `EAIFormatMessage("Data: '%2', '%3', '%1'", "A", "B", "C")`  
returns the string:

```
"Data: 'B', 'C', 'A'"
```

## Sample Siebel eScript

This section provides a sample Siebel eScript map for transforming data from a Siebel Account to SAP to retrieve an order list. The map is used to convert between the Siebel Account object and the equivalent SAP R/3 objects.

```
function GetSAPOrderStatus_SiebelToBAPI (inputMsg, outputMsg)
{
    /* Input Objects' Integration Components
    *   Order Object (Order - Get SAP Order Status (Siebel))
    *   Order
    *
    * Output Object's Integration Components:
    *   BAPI Import (Order - Get SAP Order Status (BAPI Input))
    *   Import Parameters
    */

    /*
    * Set up EAI Lookup objects
    */

    /*
    * Set up EAI Input Message objects
    */

    var iOrderObj;    // Siebel Order instance
    var iOrderComp;   // Order

    /*
    * Set up EAI Output Message objects
    */

    var oGSOBJ;       // BAPI instance
    var oGSImportComp; // Import Parameters

    /*
    * Find and create top-level integration object
    */

    iOrderObj = inputMsg.GetIntObj ("Order - Get SAP Order Status
    (Siebel)");

    oGSOBJ = outputMsg.CreateIntObj ("Order - Get SAP Order
    Status
    (BAPI Input)");

    /*
    * Read int object instances from EAI message
    */

    while (iOrderObj.NextInstance ())
    {
```

```
/*
 * Create "Get Status" object
 */

oGSObj.NewInstance ();

/*
 * Read "Order" component
 */

iOrderComp = iOrderObj.GetPrimaryIntComp ("Order");

oGSImportComp = oGSObj.CreatePrimaryIntComp
("Import Parameters");

if (iOrderComp.NextRecord ())
{
    /*
     * Write "Import Parameters" component
     */

    oGSImportComp.NewRecord ();
    oGSImportComp.SetCopySource (iOrderComp);
    oGSImportComp.CopyFieldValue ("SALESDOCUMENT",
        "Integration Id");
}
}
```



# Troubleshooting Techniques

# A

This chapter describes various troubleshooting techniques to help debugging problems you might encounter while using EAI Siebel Adapter or EAI Dispatcher.

## Service Arguments Tracing

You can export input and output arguments (property sets) in XML format to a file. This facilitates the debugging of problems with the EAI Siebel Adapter and EAI Dispatcher business services.

### **To output arguments in XML format**

- 1** Set server parameter `EnableServiceArgTracing` to `True`.

If you are running in Siebel Dedicated Web Client, add the following to your `.cfg` file:

```
[EAISubsys]

EnableServiceArgTracing = TRUE
```

This prevents the dump of the arguments when log events is set to `ALL`.

- 2** Set the appropriate event level on your server component through the server manager on the server or `SIEBEL_LOG_EVENTS` in Siebel Dedicated Web Client. One of the following events should be set.

If you set event to:

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

When arguments are output, there will be a trace log entry that indicates the file name (in the log directory).

```
EAIDispatchSvcArgTrc

EAISiebAdptSvcArgTrc
```

The filenames will take this form:

```
<service name>_<input|output>_args_<big number>.dmp
```

For example:

```
EAIDispatchService_input_args_270613751.dmp
```



# Index

## A

- ancestor components, addressing 84
- argument tracing 42
- arguments 147
  - See also *individual argument entries*
  - positional 147
  - setting the values of 102, 124
- attachments, MIME sample 127

## B

- business scenario, data mapping 85
- Business Service Simulator 42
- business services
  - See also EAI Siebel Adapter
  - arguments, setting value of 102, 124
  - data transformation, defining 90
  - EAI Dispatch Service, overview 28
  - EAI MIME Doc Converter 120
  - EAI MIME Hierarchy Converter 120
  - EAI XML Read from File 14, 76
  - errors in 144
  - mapping directly from 94
  - script entry point 92
  - transforms, role of 32

## C

- CancelOperation 96
- child components, addressing 84
- Child Hierarchy target 32
- classes. See *individual class (CSS) entries*
- Content ID, MIME messages 127
- ContinueOperation 96
- CopyAttachment method 126
- CopyFieldValue method 110, 117
- CreateIntComp method 106, 111, 118

- CreateIntObj method 102
- CreatePrimaryIntComp method 106
- CSSEAIIDTScriptService class 90
- CSSEAIIntCompIn 114
- CSSEAIIntCompOut 117
- CSSEAIIntMsgIn 98, 99
- CSSEAIIntMsgOut 98, 102
- CSSEAIIntObjIn 104
- CSSEAIIntObjOut 106
- CSSEAIMimeMsgIn 120
- CSSEAIMimeMsgOut 124
- CSSEAIPrimaryIntCompIn 107
- CSSEAIPrimaryIntCompOut
  - about 110
  - methods 110
- CSSEAIValueMap object 139, 140
- CSSEAIValueMap Translate method 140

## D

- data mapping
  - architecture diagram 62
  - business scenario 85
  - declarative and programmatic options, compared 59
  - postconditions, use of 83
  - preconditions, use of 82
- data mapping engine. See EAI Data Mapping Engine
- data maps
  - arguments, defining 67
  - component mappings, defining 68
  - creating 66
  - creation process overview 65
  - EAI Value Maps 138
  - field mappings, defining 69

- function of 90
- inbound Workflow, creating 75
- integration object maps 63
- integration object maps, viewing 66
- map types and storage locations 63
- outbound Workflow, creating 70
- performance consideration 59
- role of 63
- validating 69
- data transformation
  - data transformation business service, defining 90
  - declarative and programmatic options, compared 59
  - and empty strings 141
  - errors 144
  - functions 90
  - functions, about 97
  - integration component objects 113
  - literal strings 141
  - methods. *See individual method entries*
  - null values 141
  - options diagram 88
  - output integration objects 117
  - process overview 89
  - raising errors 142
  - Siebel Data Mapper, about 58
  - transformation function 95
  - unspecified results 141
  - untranslated values 142
  - XML property to integration object 95
- data types
  - conversion to strings 113
  - supported data types, listed 97
- data, correlating between systems 138
- debugging. *See troubleshooting*
- declarative interface. *See data transformation*
- defaultIfEmpty parameter 99, 110
- defaultIfNull parameter 99, 110
- Dispatch method 30
- dispatch rule grammar
  - example 52
  - search expression symbols 31
- dispatch service. *See EAI Dispatch Service*

- DTE business service
  - methods and method arguments 93
  - script, creating 91

## E

- EAI Data Mapping Engine
  - architecture diagram 58, 62
  - component groups, required 60
  - data map, executing 73
  - Execute method 60
  - expression categories 80
  - inbound data map, executing 77
  - methods 60
  - multiple source components, addressing 84
  - process overview 61
  - Purge method 61
  - purging a development map 76
- EAI Dispatch Service
  - See also* transforms; hierarchy, incoming; hierarchy, output
  - architecture diagram, inbound dispatch service 35
  - architecture diagram, outbound dispatch service 36
  - argument tracing 42
  - compared to Workflow 43
  - hierarchy diagram 29
  - inbound dispatch workflow, creating 38
  - inbound dispatch, business scenario 48
  - input and output, debugging 42
  - methods used 30
  - outbound dispatch, business scenario 46
  - outbound property set, business scenario 54
  - overview 28
  - process diagram 28
  - process overview 37
  - property sets, about use of 43
  - rule set, creating 39
  - rules, defining 39
  - run-time tasks 30
  - Search Expression definitions 31

EAI MIME Doc Converter Business  
     Service 120  
 EAI MIME Hierarchy Converter Business  
     Service 120  
 EAI MQSeries Server Transport 20  
     sample 20  
 EAI Siebel Adapter  
     export example query 17, 23  
     importing account information,  
         example 14  
     inbound data map, writing to  
         database 78  
     outbound data maps 72  
 EAI Value Maps  
     about 138  
     accessing 143  
 EAI XML Converter, example 20  
 EAI XML Read from File business  
     service 14, 76  
 EAI XML Write to File adapter, export  
     example 18  
 EAI XML Write to File business service 74  
 EAIDispatchSvcArgTrc 42  
 EAIExecuteMap 96  
 EAIExecuteMap function 92  
 EAIFormatMessage 147  
 EAIFormatMessage method 147  
 EAIGetValueMap 139, 141  
 EAIGetValueMap method 143  
 EAIRaiseError method 146  
 EAIRaiseErrorCode method 147  
 empty strings 141  
 EnableServiceArgTracing parameter 42  
 Error Code property 16  
 Error Message property 16  
 error symbols 147  
 errors  
     capturing 146  
     categories of 144  
     EAIFormatMessage method 147  
     raising 142  
 exception handling, in data transformation  
     scripts 144

Execute method 60, 73, 92  
 Exists method 104, 107, 114  
 export example 18

## F

fatal errors 146, 147  
 fields  
     copying 112, 117  
     setting a value for 113  
     setting values of 119  
 fields, copying 110  
 FirstInstance method 104  
 FirstRecord method 108, 114  
 formatParameters parameter 147

## G

GetArgument method 99, 121  
 GetAttachment method 101  
 GetAttachmentByCID method 101, 123  
 GetAttachmentCount method 100  
 GetFieldValue method 108, 115  
 GetIntComp method 109, 116  
 GetIntObj method 100  
 GetPart method 122  
 GetPartByCID method 122  
 GetPartCount method 122  
 GetPrimaryIntComp Method 105  
 GetXMLRootPart method 123

## H

handlers, custom written 142  
 hierarchy diagram, EAI Dispatch  
     Service 29  
 hierarchy, incoming  
     data, passing to workflow 38  
     routing methods, compared 43  
     transforms, role of 32  
 hierarchy, output  
     business service arguments, adding 33  
     Child Hierarchy target, about 32  
     Property Name target, about 33  
     root hierarchy target, about 32

- Source Expression, setting Type field to 33
  - Source Expression, setting Value field to 33
  - workflow process properties, adding to 33
- I**
- import example 14
  - inbound dispatch
    - architecture diagram 35
    - business scenario 48
    - process overview 37
    - rule set, creating 39
    - rules, defining 39
    - workflow, creating 38
  - inbound message
    - converting to business object format, example 20
    - receiving from MQSeries 20
  - #include file 92
  - input integration component, copying fields 110
  - inputPropSet 96
  - Insert method 14
  - integration component
    - creating a new 111, 118
    - primary integration component, opening for input 105
    - setting field value 119
  - integration component maps
    - about 64
    - defining 68
    - parent component field, addressing 81
    - postconditions, use of 83
    - preconditions, use of 82
    - Source Search Specifications, role of 81
  - integration field maps
    - defining 69
    - multiple source components, addressing 84
    - Source Expression example 80
    - validating 69
  - integration message objects 98
  - integration object maps
    - about 63
    - arguments, about 63
    - arguments, defining 67
    - creating 66
    - Source Search Specifications, role of 81
    - viewing 66
  - integration objects
    - about 104
    - creating a new instance of 106
    - CSSEAllIntObjIn 104
    - CSSEAllIntObjIn, methods 104
    - CSSEAllIntObjOut 106
    - new integration object, method 102
    - output 117
    - role in data mapping process 62
    - transforming from XML property set, specification 95
  - integration process, testing 25
- K**
- key values, translating 140
- L**
- letter to memo conversion, code sample 137
  - letter, converting to memo (map function) 130
  - Lookup method 30
  - lookup tables, accessing 143
- M**
- map function, converting letter to memo 130
  - MapFn function 95
  - MapName argument 92
  - memo, converting from a letter, code sample 137
  - message, inbound
    - business object format, converting example 20

- receiving from MQSeries Server
    - Transport, example 20
  - methodName 96
  - methods
    - CopyAttachment 126
    - CopyAttachmentCID 103
    - CopyFieldValue 110, 117
    - CreateIntComp 111, 118
    - CreateIntObj 102
    - CreatePrimaryIntComp 106
    - Dispatch method 30
    - EAIFormatMessage 147
    - EAIGetValueMap 143
    - EAIRaiseError 146
    - EAIRaiseErrorCode 147
    - Execute 92
    - Execute method 60, 73
    - Exists 104, 107, 114
    - FirstInstance 104
    - FirstRecord 108, 114
    - GetArgument 99, 121
    - GetAttachment 101
    - GetAttachmentByCID 101, 123
    - GetAttachmentCount 100
    - GetFieldValue 108, 115
    - GetIntComp 109, 116
    - GetIntObj 100
    - GetPart 122
    - GetPartByCID 122
    - GetPartCount 122
    - GetPrimaryIntComp 105
    - GetXMLRootPart 123
    - Lookup method 30
    - NewInstance 106
    - NewRecord 112
    - NextInstance 105
    - NextRecord 109, 116
    - Purge method 61, 72, 76
    - Read Siebel Message method 76
    - SetArgument 102
    - SetAttachmentSource 103, 125
    - SetCopySource 112, 119
    - SetFieldValue 113, 119
    - Write Siebel Message method 74
  - MIME
    - about MIME messages 127
    - Content ID 127
    - CSSEAIMimeMsgIn, object and
      - methods 120
    - CSSEAIMimeMsgOut, object and
      - methods 124
    - messages and objects, about 120
    - MIMEHierarchy argument 94
    - sample message with attachment 127
  - MIMEHierarchy argument 94
  - MQSeries
    - export example 21
    - import example 18
    - process properties, exporting
      - example 22
  - msg parameter 147
- N**
- name parameter 99, 100
  - NewInstance method 106
  - NewRecord method 112
  - NextInstance method 105
  - NextRecord method 109, 116
  - null value 108, 141
- O**
- Object Id property 16
  - outbound dispatch
    - architecture diagram 36
    - business scenario 46
    - process overview 37
    - property set, business scenario 54
  - outbound Siebel Messages 24
  - output integration component, copying
    - fields 110
  - OutputIntObjectName output
    - argument 102
  - outputPropSet 96

## **P**

- parameters. *See individual parameter entries*
- positional arguments 147
- postconditions 83
- preconditions 82
- primary integration component
  - checking existence of 107
  - creating new record 112
  - defined 107
  - field value, returning 108
  - getting for input 109
- primary output integration component
  - methods 106
- process properties
  - export in XML example 16
  - import using MQSeries queue, example 19
  - importing account information, example 13
  - inbound data map, creating 75
  - MQSeries, exporting example 22
  - outbound data maps 71
  - output hierarchy root node, adding to 33
  - passing data to workflow 38
- Property Name target 33
- property set format, functions for manipulating 129
- property sets
  - dispatch method comparison 43
  - root element in output XML, creating 130
  - root element of XML document, returning 129
  - Workflow limitation in use of 43
- Purge method 61, 72, 76

## **Q**

- Query method, EAI Siebel Adapter 23

## **R**

- Read Siebel Message method 76

- Receive method, EAI MQSeries Server
  - Transport example 20
- records, accessing next record 109
- root hierarchy target 32
- rules
  - invalid rules 53
  - rule sets, creating 39
  - rules, defining 39

## **S**

- scripts
  - See also* Siebel eScript
  - for DTE business service 91
  - exception handling 144
  - terminating on error 146, 147
- search expression grammar
  - expressions, symbols used in 31
  - sample 52
- Service\_PreInvokeMethod function, data transformation 92
- ServiceArguments 94
- SetAttachmentSource method 125
- SetCopySource method 112, 119
- SetFieldValue method 113, 119
- Siebel Data Mapper
  - about 58
  - architecture diagram 62
  - business scenario 85
  - performance 59
- Siebel Errors, in data transformation
  - scripts 144
- Siebel eScript
  - See also* scripts
  - data type conversion 113
  - including eaisiebel.js 92
  - role in data transformation functions 97
- Siebel Inbound argument 140
- Siebel Message argument 94
- Siebel Messages
  - converting outbound to XML 24
  - CSSEAIIntCompIn, object and methods 114

CSSEAIIntCompOut, object and methods 117

CSSEAIIntMsgIn integration message object 99

CSSEAIIntMsgIn object and methods 99

CSSEAIIntMsgOut object and methods 102

CSSEAIIntObjIn object and methods 104

CSSEAIIntObjOut object and methods 106

CSSEAIPrimaryIntCompIn object and methods 107

CSSEAIPrimaryIntCompOut object and methods 110

integration message objects, about 98

objects and methods, overview 98

outbound 24

Siebel Outbound argument 140

Siebel Tools, data transformation business service, defining 90

Source Expression

- example 80
- Property Name 33
- Type field, setting to 33
- Value field, setting to 33

Source Search Specification

- postconditions, use of 83
- preconditions, use of 82

sourceName parameter 110

strings, data transformation of 141

## T

targetName parameter 110

targets. *See* transforms

testing, integration processes 25

transformation maps. *See* data maps

transforms

- business services 32
- Child Hierarchy target 32
- combining transforms 33
- defining 40
- overview 32
- Property Name target 33

- root hierarchy target 32
- Type target 33
- Value target 33

troubleshooting

- argument tracing 42
- dispatch service input and output, debugging 42
- Lookup method, role of 30
- Type target 33

## U

unmapped values 141

unmappedKeyHandler

- about 140
- uses of 141

untranslated values 142

Update method 14

## V

Value target 33

## W

Workflow

- compared to EAI Dispatch Service 43
- data maps, running 92
- inbound data map, creating
  - Workflow 75
- inbound dispatch, creating 38
- integration processes, overview 11
- outbound data map, creating
  - Workflow 70
- passing incoming hierarchy to 38

workflow examples

- account information, exporting in XML 15
- account information, importing 13
- exporting using MQSeries queue 21
- importing using MQSeries queue 18

Workflow Process Simulator 25

Workflow Simulator, testing inbound
 

- Workflow 78

Write Siebel Message method 74

**X**

## XML

- dispatch input and output,
  - debugging 42
- element accessor functions 131
- exporting information, example 15
- outbound data mapping Workflow
  - example 74
- Search Expression example 52
- top-level property set functions 129
- transforming to integration object,
  - specification 95

## XML functions

- example 137
- XPSAddChild 136
- XPSCreateRootElement 130
- XPSFindChild 135
- XPSGetAttribute 133
- XPSGetChild 135
- XPSGetChildCount 134

XPSGetRootElement 129

XPSGetTagName 131

XPSGetTextValue 132

XPSSetAttribute 134

XPSSetTagName 131

XPSSetTextValue 132

XML Hierarchy argument 94

XPath standard 31

XPSAddChild function 136

XPSCreateRootElement function 130

XPSFindChild function 135

XPSGetAttribute function 133

XPSGetChild function 135

XPSGetChildCount function 134

XPSGetRootElement function 129

XPSGetTagName function 131

XPSGetTextValue function 132

XPSSetAttribute function 134

XPSSetTagName function 131

XPSSetTextValue function 132