



**SIEBEL**<sup>7</sup>  
eBusiness

## **SIEBEL PORTAL FRAMEWORK GUIDE**

*VERSION 7.5, REV. B  
AUGUST 2003*

12-FRL6RV

Siebel Systems, Inc., 2207 Bridgepointe Parkway, San Mateo, CA 94404  
Copyright © 2003 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.

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.

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

Revision History . . . . .	8
What's New for 7.5 . . . . .	9

## Chapter 1. About Siebel Portal Framework

Portal Framework Overview . . . . .	11
Portal Framework Architecture . . . . .	12
Enterprise Application Integration . . . . .	12
Portal Agents . . . . .	13
XML Web Interface . . . . .	13

## Chapter 2. Integrating External Content

Understanding Portal Agents . . . . .	15
Portal Agents and Authentication Strategies . . . . .	16
About Disposition Types . . . . .	17
Inline . . . . .	17
IFrame . . . . .	18
Web Control . . . . .	19
Form Redirect . . . . .	19
Server Redirect . . . . .	20
Portal Agent Restrictions . . . . .	21
Disposition Types Summary . . . . .	22
Task Overview for Creating Portal Agents . . . . .	23
Determining the Login Requirements . . . . .	23

Portal Agent Configuration . . . . .	27
Configuring Business Components to Handle External Data . . . . .	27
Displaying External Content Within an Applet . . . . .	28
Displaying External Content Outside of an Applet . . . . .	29
Portal Agent Administration . . . . .	29
Defining the External Host . . . . .	30
Defining Web Applications . . . . .	31
Defining Symbolic URLs . . . . .	32
Defining Symbolic URL Arguments . . . . .	35
Defining Content Fixup . . . . .	38
Defining End-User Login Credentials . . . . .	40
Example Portal Agent . . . . .	40
Review the Login Form . . . . .	41
Define the External Host . . . . .	42
Define the Symbolic URL . . . . .	43
Define Symbolic URL Arguments . . . . .	44
Define User Login Credentials . . . . .	45
Reviewing the SWE Log File . . . . .	46
Portal Agent Command Reference . . . . .	46
EncodeURL . . . . .	46
IFrame . . . . .	47
NoCache . . . . .	47
NoFormFixup . . . . .	48
PreLoadURL . . . . .	49
PostRequest . . . . .	49
UserLoginId . . . . .	50
UserLoginPassword . . . . .	50
UseSiebelLoginId . . . . .	51
UseSiebelLoginPassword . . . . .	51
WebControl . . . . .	52

### Chapter 3. Delivering Content to External Web Applications

Overview of the XML Web Interface . . . . .	53
Accessing Siebel XML . . . . .	54
Siebel OM and Web Server Configuration and Markup Determination . . .	55
Connecting to the XML Web Interface . . . . .	57
Submitting HTTP Requests Through the Web Server . . . . .	57
Submitting Requests Using the Web Engine Interface . . . . .	60
Web Engine Interface Property Names . . . . .	64
Web Engine Interface Code Sample . . . . .	67
XML Request Structure . . . . .	74
Query String . . . . .	74
XML Command Block . . . . .	74
XML Response Structure . . . . .	83
Errors . . . . .	83
XML Response . . . . .	83
HTML Response . . . . .	90
WML Response . . . . .	90
Common Operations . . . . .	90
Logging In . . . . .	90
Logging Off . . . . .	92
Navigating to a Screen . . . . .	92
Navigating Within a Screen . . . . .	93
Querying Items . . . . .	94
Modifying Records . . . . .	96
Deleting Records . . . . .	99
Picking Records . . . . .	101
SWE API . . . . .	104
SWE Commands . . . . .	105
SWE Methods . . . . .	110
SWE Arguments . . . . .	116

Document Type Definition . . . . . 119

Manipulating Siebel XML with XSL Stylesheets and XSLT . . . . . 127

    Defining SWTs Stylesheet Tags . . . . . 127

    XML-Specific Template Tag . . . . . 127

    Sample Stylesheet Code . . . . . 128

**Chapter 4. Web Engine HTTP TXN Business Service**

Scenario . . . . . 135

    Sample HTTP TXN BS Code . . . . . 137

    Sample Output by HTTP TxN BS . . . . . 141

Web Engine HTTP TXN Business Service API . . . . . 142

**Index**

# Introduction

This book provides a detailed discussion of Siebel Portal Framework. It provides overview information, tasks for integrating external content into the Siebel user interface, and reference information about the XML Web interface used to deliver content to external applications.

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>Call Center Administrators</b>	Persons responsible for setting up and maintaining a call center. Duties include designing and managing Computer Telephony Integration (CTI), SmartScripts, and message broadcasts.
<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.
<b>Siebel System Administrators</b>	Persons responsible for the whole system, including installing, maintaining, and upgrading Siebel applications.

## Product Modules and Options

This guide contains descriptions of modules that are optional and for which you may not have purchased a license. Siebel's Sample Database also includes data related to these optional modules. As a result, your software implementation may differ from descriptions in this Bookshelf. To find out more about the modules your organization has purchased, see your corporate purchasing agent or your Siebel sales representative.

# Revision History

*Siebel Portal Framework Guide*

## Version 7.5, Rev. B

**Table 1. Changes Made in Version 7.5, Rev. B**

Topic	Revision
<a href="#">“Defining Symbolic URL Arguments” on page 35</a>	In the table following step 3, corrected descriptions of the Field and Field-All Values Argument Types.
<a href="#">“Submitting Requests Using the Web Engine Interface” on page 60</a>	Replaced sample code that appears throughout this section.
<a href="#">“Web Engine Interface Property Names” on page 64</a>	Added new section.
<a href="#">“Web Engine Interface Code Sample” on page 67</a>	Added new section with sample code.

## Version 7.5, Rev. A

**Table 2. Changes Made in Version 7.5, Rev. A**

Topic	Revision
<a href="#">“Defining Symbolic URL Arguments” on page 35</a>	Corrected the descriptions of Argument Values after step two of the procedure.
<a href="#">“Adding Records” on page 95</a>	Corrected code sample for WriteRecord.
<a href="#">“Modifying Records” on page 96</a>	Corrected code sample for EditRecord and WriteRecord.
<a href="#">“SWE Methods” on page 110</a>	Corrected entries for EditRecord and WriteRecord.
<a href="#">“Web Engine HTTP TXN Business Service API” on page 142</a>	Corrected entries for GetAllRequestCookies, GetAllRequestHeaders, GetAllResponseCookies, GetAllResponseHeaders.

## What's New for 7.5

New Portal Framework features for version 7.5 are described in [Table 3](#).

**Table 3. 7.5 Portal Framework Features**

Feature	Description
Session Management and Session Re-Use	Allows the Siebel application to manage sessions that are integrated with third-party application servers, and recognizes when these sessions can be re-used, retaining the user's session context, and preventing the buildup of session overhead on the third-party application server. As these sessions time-out due to inactivity, the Siebel Server automatically re-authenticates with the third-party application server, requiring no further entry of user credentials on the part of the user.
Personalized Content Displayed Inline	Prior to version 7.5, external content delivered to a specific user (after authentication) had to be displayed in a new window. In version 7.5, the Portal Framework has been enhanced to support the display of personalized external content adjacent to other applets in the Siebel user interface.
Profile attributes sent in the HTTP request	You can configure Portal Agents to retrieve data from the Personalization business component and send it as part of the HTTP request. This allows the Portal Agent to send the external application user-specific preferences, such as the user's time zone, or preferred language.
Support for server redirects	The Siebel Web Engine now has the ability to recognize when a request sent to an external host is redirected to another host (the one that actually conducts the session). This includes handling cascaded redirects, redirects across domains, and redirects invoked by JavaScript. It allows session management to continue after a redirect occurs.
Displaying external content outside of an applet	You can now display external content outside of an applet, such as in the banner frame, using Web Page Items. Web Page Items can now be associated with a symbolic URL. Note that this feature does not apply to administering Page Items in the Portal Administration screen.

## **Introduction**

---

*What's New for 7.5*

This chapter provides an overview of the Siebel Portal Framework and summarizes the technologies that make up the Portal Framework.

## Portal Framework Overview

Enterprises are often composed of many different information technology resources, such as:

- Shared network directories.
- Department intranet sites.
- Legacy applications.
- Applications developed in-house.
- Purchased Web applications.

With many disparate applications and technologies, IT resources are difficult to maintain and difficult to use. For example, applications:

- Follow different user interface guidelines.
- Are rendered with different themes.
- Track profile attributes differently.
- Vary in the quality of online assistance.
- Have separate login and password credentials.
- Have different search functionality.

One solution to this problem is to integrate the various applications and content sources used in an enterprise and present them in a single user interface, called a portal. The Siebel Portal Framework allows you to do this. The Portal Framework provides you with the tools and supporting technologies that allow you to:

- Aggregate external data with Siebel data and present it in the Siebel user interface.
- Deliver Siebel data to external applications.
- Integrate external application business logic and data with Siebel applications.

## Portal Framework Architecture

The portal framework includes the following framework components:

- Enterprise Application Integration
- Portal Agents that integrate external content into the Siebel user interface
- XML Web interface for delivery of Siebel content to external applications

## Enterprise Application Integration

Siebel EAI provides mechanisms for sharing data and business logic with other applications, including:

- Integration Objects
- Virtual Business Objects
- Programming APIs
- Predefined adapters and connectors

For more information about Siebel EAI, see *Overview: Siebel eBusiness Application Integration Volume I* and other EAI titles on the *Siebel Bookshelf*.

## Portal Agents

Portal Agents provide you with a mechanism to retrieve content from a non-Siebel source and display it in the Siebel user interface. The Portal Agent retrieves content on behalf of the user, logging on to the external application using the user's credentials and retrieving only the content that is targeted for the user. Portal Agents provide single sign-on capability and a profile tracking mechanism.

See [“Understanding Portal Agents” on page 15](#) for more information about Portal Agents.

## XML Web Interface

In enterprises where a non-Siebel portal framework is already established, you need to be able to deliver Siebel content to other applications and frameworks. The XML Web interfaces provides you with a mechanism to deliver Siebel data to external applications as XML documents. This provides the external application with a flexible format for integrating Siebel data into its user interface.

See [Chapter 3, “Delivering Content to External Web Applications,”](#) for more information.

## **About Siebel Portal Framework**

*Portal Framework Architecture*

This chapter provides an overview of Portal Agents. It covers the configuration and administration tasks necessary to display external content in the Siebel user interface. It also includes a reference section that lists all the commands available for use with Portal Agents.

## Understanding Portal Agents

Portal Agents allow you to integrate external data into the Siebel user interface. Portal Agents retrieve data by sending HTTP requests to external applications, and then display the HTML results in a Siebel applet or on some other portion of a Siebel Web page.

Portal Agents combine a set of features and technologies that allow you to integrate external content at the user interface layer, including:

**Single Sign-On technology (SSO).** For applications that are participating in a single sign-on framework, this feature eliminates the need for the user to enter login credentials, such as username and password, more than once per work session.

**Session Management and Session Reuse.** Allows the Siebel application and the external application to maintain a user's session context, without reauthenticating for subsequent requests. This minimizes session resource overhead on the external application, and allows the user to retain session context, such as Shopping Cart contents.

**Time-Out Handling.** The Siebel Server automatically reauthenticates when a request is submitted after the external application's timeout period has passed.

**Symbolic URLs, with Multiple Disposition Types.** Allows content to be displayed in different ways, such as in a new browser window, inline with the other content, in an `<iframe>` tag, or as an ActiveX object embedded in the Siebel application Web page. See [“About Disposition Types” on page 17](#) for more information.

**Session Proxy.** For content integrated using a disposition type of Inline, the Siebel Server manages the interactions with external applications on behalf of the user. For more information about the Inline disposition type, see [“Inline” on page 17](#).

**Symbolic URL Commands.** Commands that direct the Portal Agent to assemble the URL for the external application in a number of ways. These include dynamically referencing the user's login and password, retrieving stored login and password values, retrieving data from the user's personalization profile, establishing the size of an `<iframe>` tag, and determining whether to set the browser cookies from the application server's login page. For a complete list of commands, see [“Portal Agent Command Reference” on page 46](#).

---

**NOTE:** Portal Agents do not integrate data at the data layer or integrate business logic. Other mechanisms in the Siebel Portal Framework, such as Integration Objects and Virtual Business Components, are designed to meet those types of integration needs. See *Overview: Siebel eBusiness Application Integration Volume I* for more information about EAI.

---

## Portal Agents and Authentication Strategies

Portal Agents can be configured to support different authentication strategies:

- **Simple Portal Agents.** External application does not require any authentication parameters.
- **Single Sign-On Portal Agents.** External application requires authentication parameters.
  - NCSA-basic Portal Agents send a username and password as part of the URL in plain text. NCSA is no longer widely used as an authentication mechanism.
  - Form-based Portal Agents send authentication parameters as part of the body portion of the HTTP request.

## About Disposition Types

One of the steps in setting up a Portal Agent is creating a Symbolic URL. The Symbolic URL specifies the information necessary to construct the HTTP request to send to the external application. Symbolic URLs can be one of several disposition types. The disposition type determines:

- The interaction between the browser, the Siebel Server, and the external application.
- How external content is displayed in the user interface.

It is important to understand these disposition types and determine the one that suits your integration needs. Each disposition type is discussed in one of the following sections:

- [“Inline” on page 17](#)
- [“IFrame” on page 18](#)
- [“Web Control” on page 19](#)
- [“Form Redirect” on page 19](#)

The procedure for defining Symbolic URLs is covered in [“Defining Symbolic URLs” on page 32](#).

### Inline

With a symbolic URL disposition type of Inline, the Siebel Server receives content sent by an external application. It combines the external content with Siebel-produced content and composes a single HTML page, which it then sends to the client browser for display to the user. Optionally, links in the aggregated content are rewritten so they reference the Siebel Server (proxy), rather than referencing the external application server directly. This allows the Siebel Server to handle links in the aggregated content so that from the user’s perspective the content comes from one integrated application rather than from different application servers.

The inline disposition type supports Session Management. Session Management is a feature that allows the Siebel Server to manage session cookies and allows it to automatically relogin to an external application after a time out occurs.

Although the Inline disposition type is the preferred disposition type, it will only work in rare cases. The inline disposition type is a good option when the page you are trying to integrate is a simple HTML page with simple JavaScript. If the page you are trying to integrate has complex JavaScript or references frames, then the Inline disposition type will not work and you should try the IFrame disposition type. The Inline disposition type supports the GET method only. Also, the number of characters that can appear in the URL is limited to 2048 characters.

## IFrame

Use this disposition type when aspects of the external application do not allow content to be aggregated with other Siebel content. See [“Portal Agent Restrictions” on page 21](#) for more information about when this may occur.

The IFrame disposition type uses the `<iframe>` tag to create an Internal Frame as part of the page generated by the Siebel Server. It allows the Portal Agent to retrieve content to populate the Internal Frame. This content does not pass through the Siebel Server, but is directly requested by the client and sent by the application server to the user's browser. Although this disposition type is not as preferable as the Inline disposition type, in most cases, it is the method that works.

The IFrame disposition type supports JavaScript and Frames. Therefore, if the Inline disposition type does not work, the IFrame option is the best option. The IFrame disposition type also supports the Session Keep Alive feature. However, it does not support Session Management.

The IFrame disposition type will work in many cases. However, it does not work when frames displayed within the `<iframe>` tag refer to top-level JavaScript objects. If frames in the page you are trying to integrate refer to top-level JavaScript objects, then try the Web Control disposition type.

---

**NOTE:** The IFrame disposition type is supported on Internet Explorer 5.5 and above.

---

## Web Control

Use the Web control disposition type when IFrame or Inline disposition types do not work. Typically this is because of hardcoded references to specific frame names in the external application's HTML. See [“Portal Agent Restrictions” on page 21](#) for more information.

The Web Control disposition type embeds an Internet Explorer ActiveX object in the Siebel page and provides it to the external application. In the Web Control disposition type, similar to the IFrame type, the external application sends content directly to the user's browser, bypassing the Siebel Server. The external application then behaves as if the ActiveX IE instance is an independent Web browser.

---

**NOTE:** The Web Control disposition type is supported for Internet Explorer 4.0 and above.

---

## Form Redirect

The Form Redirect disposition type is not commonly used with Siebel eBusiness applications, version 7.5.

In the Form Redirect scenario, the Siebel Web client submits a request to the Siebel Server. The Siebel Server creates a form with the necessary authentication information in it, and then sends the form back to the browser. The browser loads the form and then submits it to the external host for processing. The external host sends back the results, which the browser displays in a new window.

The Form Redirect disposition option is usually displayed in a new window, rather than inline with other Siebel applets.

### Server Redirect

The Server Redirect disposition type is no longer used in version 7.5, but is included in this guide for customers running prior versions of Siebel eBusiness applications.

In the Server Redirect scenario, the browser sends a request to Siebel Server. The Siebel Server sends the authentication request to the external application. After it receives a response from the external application, it creates a 302 Response, which contains the value of the target URL in the header. The Siebel Server sends the 302 Response back to the browser and the browser is redirected to the target host.

Like Form Redirect, Portal Agents that use the Server Redirect disposition type should be configured to display the results in a new window rather than inline with other Siebel applets.

When using the Server Redirect disposition type, there is one required argument and several optional arguments. The arguments are defined below:

- **SSO\_REDIRECT\_PATH.** This is a required argument that defines the URL to which the browser is to be redirected. The value of this argument is sent as the URL in the header of the 302 Response.
- **SSO\_COOKIE\_DOMAIN.** This is an optional argument that specifies a domain for which the cookie is valid. If this argument is not defined, the domain of the external host (as defined in Host Administration view) is used.
- **SSO\_COOKIE\_NAMES.** This is an optional argument that allows you to define the cookies to be sent to the browser from the external application. If no cookie names are defined, the Portal Agent sends all cookies from the external site to client browser. For example, the argument defined in [Table 4](#) specifies that only the cookies named *Pid* and *tid* are to be sent to the browser from the external host.

**Table 4. Example Symbolic URL Argument**

Name	Argument Type	Argument Value
SSO_COOKIE_NAME	Constant	Pid;tid

- **SSO\_COOKIE\_PATH.** This is an optional argument that allows you to define a subset of URLs to which the cookies apply.

You define these arguments in the Symbolic URL Argument applet. See [“Defining Symbolic URL Arguments” on page 35](#) for instructions on how to do this.

## Portal Agent Restrictions

Portal Agents are meant to bring existing applications and content into the Siebel user interface without requiring additional modifications of the external application. However, this is not always possible due to the way HTML and Web browsers are designed. For example:

- The use of frames by an external application may not be amenable to inline aggregation methods.
- Specific frame references in the returned content referring to global frames (`_NEW`, `_TOP`, `.parent()`) may not be amenable to inline aggregation methods.
- Reliance on JavaScript functions defined in (assumed) external frames may not be amenable to inline aggregation methods.
- URLs that are created dynamically by JavaScript may not be amenable to any fixup techniques, as the URLs would not be easily parsed on the HTML content.
- `<head >` sections of Web pages are normally stripped by fixup code. If `<head >` sections contain JavaScript functions definitions, these may be stripped by fixup code and dependent JavaScript would not run properly.

For these reasons, an Inline disposition type does not work often. However, if you control both the Siebel application instance as well as the external application, and can resolve some of these issues, you should be able to get the Inline disposition type to work correctly. For more information about the Inline disposition type, see [“Inline” on page 17](#).

If you do not have control over the external application, the IFrame disposition type is the most likely method that will provide satisfactory results. It works with about 80% of the form-based application sites tested. For more information about the IFrame disposition type, see [“IFrame” on page 18](#).

## Disposition Types Summary

Table 5 summarizes the characteristics of each disposition type.

**Table 5. Disposition Type Summary**

Disposition Type	Benefits	Drawbacks
Inline	<ul style="list-style-type: none"> <li>■ Inline integration into the Siebel user interface.</li> <li>■ Session Management, including managing session cookies and automatic re-login after time out.</li> </ul>	<ul style="list-style-type: none"> <li>■ Only works in very few cases.</li> <li>■ Will not work with complex JavaScript.</li> <li>■ Will not work if there are reference to frames.</li> <li>■ Supports the GET method only.</li> <li>■ URL limited to 2048 characters.</li> </ul>
IFrame	<ul style="list-style-type: none"> <li>■ Inline integration into the Siebel user interface.Supports complex JavaScript.</li> <li>■ Supports references to frames.</li> <li>■ Session Keep Alive supported.</li> <li>■ Works for most cases.</li> </ul>	<ul style="list-style-type: none"> <li>■ No session management.</li> <li>■ Only supported by IE5.5 and higher.</li> <li>■ Does not support frames that reference top-level JavaScript objects.</li> </ul>
Web Control	<ul style="list-style-type: none"> <li>■ Supports frames that reference top-level JavaScript Objects, because JavaScript does not refer to objects outside of the Web control.</li> </ul>	<ul style="list-style-type: none"> <li>■ No session management.</li> <li>■ Browser functionality, such as the back button, is only available by right-clicking in the Web control.</li> <li>■ ActiveX objects that contain other objects are reset if you change tabs and then return to the Web control.</li> <li>■ Web control requires more system overhead than IFrame.</li> <li>■ Only supported for IE4 and higher.</li> </ul>

## Task Overview for Creating Portal Agents

To create a Portal Agent, the following tasks are required:

- 1 [Determining the Login Requirements on page 23.](#)
- 2 [Configuring Business Components to Handle External Data on page 27.](#)
- 3 Complete one of the following:
  - [Displaying External Content Within an Applet on page 28.](#)
  - [Displaying External Content Outside of an Applet on page 29.](#)
- 4 [Defining Web Applications on page 31.](#)
- 5 [Defining Web Applications on page 31.](#)
- 6 [Defining Symbolic URLs on page 32.](#)
- 7 [Defining Symbolic URL Arguments on page 35.](#)

## Determining the Login Requirements

Before you configure Portal Agents, you need to understand what information is required by the external application to authenticate users. Typically this information is gathered using a form page, also called a login page, and then sent to the external application. You must determine exactly what information the form gathers from the user and sends to the external application, including field names and values.

In cases where you have specific knowledge about how an external application is implemented and can consult with authoritative sources regarding how the application authenticates users, determining the required input fields and values is relatively simple.

In cases where you do not have specific knowledge about how an external application is implemented, you must attempt to understand its authentication method by examining the application's login page. The steps below describe an approach that you can use to reverse engineer a login page and provide related Portal Agent configuration tips.

---

**NOTE:** It is not always possible to reverse engineer a login page. For example, JavaScript may process login field values prior to delivering the POST back to the application server, session values may be encoded in the form itself, or session values may be stored in the browser's session cookies.

---

#### **To reverse engineer a login page**

- 1** Navigate to the external application's login page and determine whether the external application uses NCSA Basic Authentication or Form-based authentication.

NCSA Basic is an older authentication mechanism that requires the browser to prompt the user for login name and password before displaying the page. NCSA Basic provides rudimentary protection against trespassers. Usually, you can tell if a site is using NCSA basic because a small popup dialog box will appear asking for login credentials. If this is the case, you can configure the Portal Agent using NCSA basic as the authentication method. See [“Defining Symbolic URLs” on page 32](#) for more information.

- 2** If the external application uses form-based authentication, view the login page's HTML using your browser's view source command.
- 3** Identify the form on the login page that asks for user credentials (often the form will ask for other information as well) and identify the input fields in this form used to authenticate users.

It is usually best to strip out all non-form lines of HTML and to isolate the `<input>` tags. That is, remove lines previous to `<form...>` and after `</form>` and remove lines not part of the `<input>` tags.

**4** Determine if the method attribute of the form tag is `POST`.

If it is `POST`, you will need to define the `PostRequest` command as an argument of the symbolic URL. See [“Defining Symbolic URL Arguments” on page 35](#) and [“PostRequest” on page 49](#) for more information.

If it is `GET`, you do not need to define a symbolic URL command, because the default method of symbolic URLs is `GET`.

**5** Determine the target of the form’s action attribute, which is usually specified as `action = "some string"`.

If the target of the action attribute is an absolute URL, one that begins with `http`, or a forward slash (`/`), use this URL as the base of the Portal Agent.

If it is a relative address, you also need to determine where the root of the URL is defined. It could be defined relative to the URL of the login page itself (most common), in a `<codebase>` tag (rare), or in JavaScript (hard to determine).

The target URL is defined using the Host Administration View and the Symbolic URL Administration view. See [“Defining the External Host” on page 30](#) and [“Defining Symbolic URLs” on page 32](#) for more information.

**6** Determine any argument values defined in the target URL.

These are the characters after the `“?”` character. Usually these are simple field-value constants. The exception is when a field or a value is a session identifier that is dynamically assigned by the external application server and is only valid for a period of time before it times out. In this case, it may not be possible to configure a Portal Agent.

You will define any argument values contained in the target URL as symbolic URL arguments. See [“Defining Symbolic URL Arguments” on page 35](#) for more information on how to do this.

- 7 Identify each of the form's `<input>` tags and determine which ones are necessary to send to the external application for authentication.

Often there are `<input>` tags in the form with a `type` attribute of `hidden` that are not evident when interacting with the application. Determining whether hidden fields are optional or required is often process of trial and error.

Some `<input>` tags will not have values identified. Either these fields are awaiting input to be entered by the user (for example, login name or password), or they are hidden fields with no values.

- If the input field is specific to the user (it asks for the user's login name and password), you can use `UserLoginId` and `UserLoginPassword` commands to instruct the Portal Agent to retrieve the user's credentials from the user's My Logins view. See [“Defining End-User Login Credentials” on page 40](#) for more information.
- If there are hidden fields with no values, when you enter them as symbolic URL arguments, make sure that the Required Argument column is not checked. If it is checked, and the input field has no value, the Portal Agent will not send this request to the target application server because there is no value to put in its place.

You will need to define the input fields and values as symbolic URL arguments. See [“Defining Symbolic URL Arguments” on page 35](#) for more information.

---

**NOTE:** The Mozilla browser includes a page info command (^I) that analyzes forms on a page and displays the method, input fields, and so on.

---

## Portal Agent Configuration

Using Portal Agents to integrate external content into the Siebel user interface requires some simple configuration in Siebel Tools. You must configure a field on the business component to handle external data and then configure either an applet or a Web page item to display the content in the user interface. An applet displays external content inside the applet container on a view. A Web page item displays external content outside of an applet, such as in the banner frame for example.

---

**NOTE:** This section describes the configuration tasks that are unique to integrating external content with the Siebel user interface. It does not describe standard configuration tasks that you may be required to perform. For example, after you configure an applet to display external content, you may have to associate that applet with a view, add the view to a responsibility, and so on. These additional tasks are standard procedures for configuring Siebel applications and are outside the scope of this book. For more information about configuring Siebel applications, see *Siebel Tools Reference*.

---

### Configuring Business Components to Handle External Data

To configure business components to handle external data using a Symbolic URL, you need to create a new calculated field on the business component. Rather than representing structured content, such as records in a database, this field will represent the HTML content sent from an external host.

---

**NOTE:** Although a symbolic URL displays data that is not stored in the database, the business component must have at least one record stored in an underlying table so that it is instantiated at run time.

---

#### **To configure a business component to handle external data using a Symbolic URL**

- 1 Create a new field on the business component.
- 2 Set the field's Calculated property to TRUE.
- 3 Set the field's Type property to DTYPE\_TEXT.

- 4 In the Calculated Value field, enter the name of the Symbolic URL (enclosed in double quotes) that you want to use to submit the HTTP request.

The name of the symbolic URL in the Calculated Value field must be enclosed in double quotes so that it evaluates as a constant.

See the business component named *AnalyticsSSO* in the Siebel Repository for an example of fields configured this way.

## Displaying External Content Within an Applet

After you have created the calculated field on the business component, you need to expose it in the user interface. You display the external content using a control in a form applet or list applet.

---

**NOTE:** You can also expose external content outside an applet, such as in the banner area. See [“Displaying External Content Outside of an Applet” on page 29](#).

---

### **To display external content within an applet**

- 1 Create an applet that you want to use to display the external content.

The applet must be based on the business component that you configured in [“Configuring Business Components to Handle External Data” on page 27](#).

- 2 Add a new control or list column to the applet.
- 3 Associate the control or list column with a calculated field on the business component that is configured to represent the external data.
- 4 Set the control or list column’s Field Retrieval Type property to *Symbolic URL*.
- 5 Set the control or list column’s HTML Type property to *Field*.

## Displaying External Content Outside of an Applet

After you have created the calculated field on the business component, you need to expose it in the user interface. You can display the external content outside of an applet using Web Page Items.

---

**NOTE:** You can also expose external inside an applet using an Applet Control or List Column. See [“Displaying External Content Within an Applet” on page 28](#).

---

### **To display content outside of an applet**

- 1 Go to the Web Page object type and select the Web page on which you want to display external data.
- 2 Create a new Web Page Item or use an existing one.
- 3 Set the Type property of the Web Page Item to *Field*.
- 4 Create the following two Web Page Item Parameters:

Name	Value
FieldRetrievalType	Symbolic URL
SymbolicURL	[ <i>name of symbolic URL</i> ]

---

**NOTE:** The Symbolic URL is mapped to the calculated field defined for the business component.

---

## Portal Agent Administration

You administer Portal Agents through several views located under the Integration Administration Screen in the Siebel Web client. These views allow you to define how links should be handled, define the external host, and define the HTTP request that is sent to the external host.

### Defining the External Host

You define the external data hosts in the Host Administration view. This view allows you to:

- Maintain external host names in a single place.
- Specify NCSA Basic authentication credentials.
- Define how links should be handled (fixed-up) after external HTML content is rendered.

#### **To define a data host**

- 1 Navigate to Site Map > Integration Administration > Host Administration view.
- 2 Enter a new record and define the necessary fields.

Some of the fields are described in the following table:

Field	Comments
Name	Name of the external host.
Virtual Name	User-defined name for the host.
Authentication Type	Select NCSA Basic if the external application requires username and password values sent in plain text in the request header. Leave this value blank in either of the following cases: <ul style="list-style-type: none"><li>■ The external application has no authentication requirements.</li><li>■ The external application uses form-based authentication and thus requires authentication arguments sent in the header or body of the request. Arguments to be sent along in the request are defined using the Symbolic URL Arguments applet. See <a href="#">“Defining Symbolic URLs” on page 32</a> for more information.</li></ul>
Authentication Value	Enter the values required for NCSA Basic authentication.

## Defining Web Applications

Web applications allow multiple Symbolic URLs to send requests to the same Web application and share the same session. This is useful if you have two different applet controls that use Symbolic URLs to submit requests to the same Web application. You can associate these Symbolic URLs to a single Web application and define whether or not they should share the same session.

There may be cases in which you do not want requests to share the same session. For example, you may not want to share a session when a session cookie contains more information than the session ID, as this could result in unexpected behavior. When you define a Web application, you specify whether or not it should share sessions.

Web applications also allow you to define the Time Out value for the session time out feature. The Session Time Out feature is only applicable to Symbolic URLs with a Disposition type of Inline.

### **To define a Web application**

- 1 Navigate to Site Map > Integration Administration > Web Application.
- 2 Enter a record and complete the fields.

Some fields are described in the following table:

<b>Field</b>	<b>Description</b>
Shared	Indicates whether or not requests generated by Symbolic URLs associated with this Web application share the same session.
Time Out	Defines the time out parameter for the Session Management feature, which is only applicable to Symbolic URLs with a disposition type of Inline.

### Defining Symbolic URLs

You use the Symbolic URL Administration view to specify how the HTTP request to the external application should be constructed and to define any arguments and values to be sent as part of the request.

#### To define a Symbolic URL

- 1 Navigate to Site Map > Integration Administration > Symbolic URL Administration.
- 2 In the Symbolic URL Administration view, enter a new record.

Some fields are defined in the following table:

Field	Description
URL	<p>Use the URL field to enter a URL for the external application. A best practice is to substitute the host's Virtual Name, the one that you defined in the Host Administration view, for the host's actual name. Doing this makes administering host names easier, because you may have many symbolic URLs pointing to one host. If the host name changes, you only need to change it in the Host Administration applet rather than having to change it in several Symbolic URL definitions.</p> <p>For example, <code>https://Virtual_Host/path...</code></p> <p>For applications that use form-base authentication, the URL is identified by the action attribute of the Form tag. See <a href="#">"Determining the Login Requirements" on page 23</a> for more information.</p>
Host Name	<p>The Virtual Name of the host defined in the Host Administration view.</p>

Field	Description
Fixup Name	<p>Name of the fixup type defined in the Fixup Administration view. The fixup type defines how links embedded in the external HTML content are rendered. For example:</p> <p><b>Default.</b> Use this fixup type with the IFrame disposition type. Link fixup is inside the view. This fixup does not affect any of the links. The links (relative or absolute) remain as they are with the content being passed back in its original form.</p> <p><b>InsideApplet.</b> This fixup converts all of the relative links to absolute links and any links using a host defined in the Host Administration view are proxied in order to maintain SWE context.</p> <p><b>OutsideApplication.</b> This fixup converts all of the relative links to absolute links using the host and path of the parent URL. No links are proxied.</p>
Multivalue Treatment	<p>Determines how arguments are handled. Possible values are:</p> <p><b>Comma Separated.</b> Instructs SWE to insert a comma between the values defined in the Symbolic URL arguments when appending the arguments to the URL. It inserts a comma after the value in the first Argument Value field and the first value in the second Argument Value field. The second Argument Value field is simply a text string entered by the user.</p> <p><b>Separate Arguments.</b> Instructs SWE to enter separate arguments for each value defined in the two Argument Value fields.</p> <p><b>Use First Record Only.</b> Uses the first record in the current record set.</p>

Field	Description
SSO Disposition	<p>The value selected in this field determines how the HTTP request is constructed and sent and how the external content is rendered in the user interface. Possible values are:</p> <ul style="list-style-type: none"><li>■ <b>Inline.</b> Proxies the request through the Siebel Server and displays content inline with other applets on a view.</li><li>■ <b>IFrame.</b> Uses the <code>&lt;iframe&gt;</code> tag to display content inline with other applets on a view.</li><li>■ <b>Web Control.</b> Uses an ActiveX control to display content inline with other applets on a view.</li><li>■ <b>Form Redirect.</b> SWE constructs a form which it sends back to the browser, which the browser then sends to the external host. The content received is displayed in a new window.</li><li>■ <b>Server Redirect.</b> SWE sends the browser a 302 Response with the value of the external host's URL in the header. The browser is redirected to the external host. The content received is displayed in a new window. Note that for Server Redirect there is a required Symbolic URL argument. See <a href="#">“Server Redirect” on page 20</a> for a description.</li></ul> <p>See <a href="#">“Understanding Portal Agents” on page 15</a> for detailed descriptions of each Disposition Type.</p>
Web Application	<p>Associates a Web Application with this Symbolic URL. For more information about Web Applications, see <a href="#">“Defining Web Applications” on page 31</a>.</p>

## Defining Symbolic URL Arguments

Symbolic URL Arguments allow you to configure Portal Agents in several ways. You use Symbolic URL Arguments for two purposes, to define data to be sent to an external host and to submit commands to SWE that affect the behavior of Portal Agents.

When defining arguments that send data, such as authentication requirements, the Argument Name and Argument Value are appended to the URL as a attribute-value pair. You can define symbolic URL arguments that send data as constants or that dynamically retrieve data from the Siebel database. Symbolic URLs allow you to retrieve data from the user's instantiated Siebel business component, such as Service Request or Account, or retrieve data from the Siebel Personalization business component, such as the user's ZIP Code or Language.

---

**NOTE:** See [“Determining the Login Requirements” on page 23](#) for information about how to determine required data for applications that use form-based authentication.

---

Symbolic URL Arguments also allow you to implement commands which you use to define the behavior of Portal Agents. See [“Portal Agent Command Reference” on page 46](#) for usage descriptions of available commands.

### **To define Symbolic URL Arguments**

- 1** Navigate to Site Map > Integration Administration > Symbolic URL Administration.
- 2** In the Symbolic URL Administration applet, select the Symbolic URL you want to configure.

- 3 In the Symbolic URL Arguments applet, enter the arguments that need to be sent to the external host.

Some of the fields are defined in the following table:

Field	Description
Name	<p>Name of the argument. For arguments of type Constant, Field, and Personalization Attribute, this field defines the exact field name expected by the external application. It is the first part of a attribute-value pair appended to the URL.</p> <p>For argument types of commands, the Name can usually be anything. The only exception to this is for the EncodeURL and PreloadURL commands. See <a href="#">“Portal Agent Command Reference” on page 46</a>.</p>
Required	<p>When this field is checked (default) the argument must have a value. If you are configuring an argument that does not have a value, uncheck the Required field. If an argument has no value and the Required field is checked, the request is not sent because there is no value to append to the URL.</p>
Argument Type	<p>They Argument Type determines the source of the data to be send along in the HTTP request. Possible values are:</p> <p><b>Constant.</b> Sends the value defined in the Argument Value field in the request.</p> <p><b>Field.</b> Sends the value of a single-value or multi-value field from the current Siebel business component.</p> <p><b>Personalization Attribute.</b> Sends the value of a field from the Personalization business component.</p> <p><b>URL Argument.</b> Data comes from the named argument of the current request.</p> <p><b>Language Value.</b> The user’s current language setting; for example, ENU.</p> <p><b>Command.</b> Implements commands that allow you to affect the behavior of the symbolic URL. For a complete list of commands see <a href="#">“Portal Agent Command Reference” on page 46</a>.</p> <p><b>Field - All Values.</b> Data from all records in the working record set for the current business component are sent in the request. This argument type is only valid for eContent Services functionality, such as multiple stock ticker retrieval.</p>

Field	Description
Argument Value	<p>The value of the argument varies depending on the Argument Type. Descriptions of possible values for each argument type are described below.</p> <p>If the Argument Type is:</p> <ul style="list-style-type: none"> <li>■ Constant, the Argument Value is the second part of the attribute-value pair that is appended to the URL.</li> <li>■ Field, the Argument Value defines a field name from the current business component. The data from this field is the second part of an attribute-value pair that is appended to the URL.</li> <li>■ Profile Attribute, the Argument Value defines a field name on the Siebel personalization business component. The data from this field is the second part of an attribute-value pair that is appended to the URL</li> <li>■ URL Argument, the Argument Value defines the name of the argument on the incoming SWE request.</li> <li>■ Language Value, the Argument Value is left null.</li> <li>■ Command, the Argument Value typically defines the name of the command. See <a href="#">“Portal Agent Command Reference” on page 46.</a></li> <li>■ Field - All Value, the Argument Value defines the field name on the current business component from which data is to be retrieved.</li> </ul>
Argument Value	Although this field is rarely used, it can be used to identify additional arguments.
Append as Argument	When this field is checked (default), the value is added as a URL argument on the outgoing request. If this field is not checked, the value will be substituted in the text of the outgoing URL.
Sequence	Determines the sequence of the arguments. In some cases the target host requires arguments in a particular order.

---

**NOTE:** If the disposition type of the symbolic URL is Server Redirect, then there is a required argument and several optional arguments that provide the data necessary for SWE to construct the header in the 302 Response. See [“Server Redirect” on page 20](#) for the details.

---

### Defining Content Fixup

The Fixup Administration view allows you to define how links embedded within external HTML content should be rendered in the Siebel user interface. The fixup types you define here will be associated with Symbolic URLs.

**To define a fixup type**

- 1 Navigate to Site Map > Integration Administration > Fixup Administration.

## 2 Enter a new record and define the fields.

Some of the fields are described in the following table:

Field	Comments
Link Context	<p>Select one of the following values:</p> <ul style="list-style-type: none"> <li>■ <b>Do Nothing.</b> This fixup does not affect any of the links. The links (relative or absolute) remain as they are with the content being passed back in its original form.</li> <li>■ <b>Outside Application.</b> This fixup converts all of the relative links to absolute links using the host and path of the parent URL. No links are proxied.</li> <li>■ <b>Inside Application.</b> This fixup converts all of the relative links to absolute links and any links using a host defined in the Host Administration view are proxied in order to maintain SWE context. After the user clicks a link, this fixup type renders HTML in the view, using the entire view for display.</li> <li>■ <b>Inside Applet.</b> This fixup handles links the same way as the Inside Application fixup type. However, in this case, when a user clicks a link, it renders HTML within an applet. The other applets remain present on the view.</li> </ul>
Context View Name	Name of view that will display the link. This is optional.
Link Target	Specifies the name of a specific target frame of the link. For example, “_blank” for a new browser window or “ <i>AnyName</i> ” to open a window of that name. This option is not often used.

---

**NOTE:** Fixup is required for all links within high-interactivity applications.

---

# Defining End-User Login Credentials

The Portal Framework provides a mechanism to store user login credentials for external Web applications. The SSO Systems Administration view allows you to specify an external application and then enter login credentials on behalf of users. The My Logins view, located in the User Preferences screen, is used by end users to maintain their own credentials.

### **To specify an external Web application and define login credentials**

- 1 Navigate to View > Site Map > Integration Administration > SSO Systems Administration.
- 2 In the SSO Systems list, enter a new record and define the following:

Field	Description
System Name	Name of the external Web application.
Symbolic URL Name	Select the name of the Symbolic URL that interacts with the external Web application.  The symbolic URL must be configured with the <a href="#">UserLoginId</a> and <a href="#">UserLoginPassword</a> commands as arguments. These arguments instruct the symbolic URL to pass the stored login credentials when authenticating with an external Web application.
Description	Enter a description of the Web application.

- 3 If you are defining login credentials on behalf of end users, in the SSO System Users list, enter end-user login names and passwords.

## Example Portal Agent

This section provides an example of using a symbolic URL to integrate content from Siebel.COM. The high-level steps for doing this are:

- 1 [Review the Login Form.](#)

- 2 Define the External Host.
- 3 Define the Symbolic URL.
- 4 Define Symbolic URL Arguments.
- 5 Define User Login Credentials.
- 6 Test.

Each of these steps is covered in the following sections.

---

**NOTE:** This example assumes the underlying objects are already configured to support the symbolic URL. See [“Portal Agent Configuration” on page 27](#) for information on how to do this.

---

## Review the Login Form

By reviewing the login page at [www.siebel.com](http://www.siebel.com), you can determine the target URL of the Action attribute and the required arguments that are being passed to the Web application. The login page at [www.siebel.com](http://www.siebel.com) contains the following `<form>` tag and `<input>` tags:

```
<form action="/index.shtm" method="POST" name="frmPassLogin"
onsubmit="return logincheck();">

  <input TYPE="TEXT" NAME="SearchString" SIZE="18"
MAXLENGTH="100" VALUE=""

  <input type="hidden" value="All" name="sc">

  <input type="hidden" value="ON" name="FreeText">

  <input type="image" src="/images/nav/button/
bttn_form_arrow.gif" NAME="Action" border="0" / alt="Submit
Search"></td>

  <input type="text" name="username" size="18">

  <input type="password" name="password" size="18">

  <input type="image" src="/images/nav/button/
bttn_form_arrow.gif" border="0" name='login' />
```

```
<input type="checkbox" name="remember" checked/>&nbsp;  <span  
class="bdDkGray">Remember my Login<br></span  
  
</form>
```

From the `action` attribute of the form tag you can determine that the target URL is relative to the root of the login page's URL. Therefore the target URL is:

```
www.siebel.com/index.shtm
```

You can also determine that the `method` attribute of the form tag is post:

```
method="POST"
```

After reviewing the `<input>` tags, you can determine that the required arguments are:

```
username  
  
password
```

---

**NOTE:** Notice that not all input fields are necessary for login.

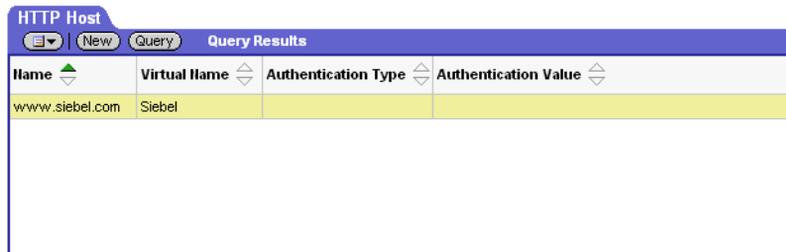
---

For more information about reviewing login forms, see [“Determining the Login Requirements” on page 23](#).

## Define the External Host

The external host is simply the address of the login page. In this example it is `www.siebel.com`. Be sure to provide a meaningful name in the Virtual Host Name field. This value is used when defining the Symbolic URL definition rather than the actual host name. This will make administration easier if the host name changes. Also notice that there is no value for the Authentication Type. A value is necessary only when using NCSA basic authentication.

Figure 1 shows the external host defined for this example.



The screenshot shows a web application window titled "HTTP Host" with a "Query Results" tab. The table below is the result of a query.

Name	Virtual Name	Authentication Type	Authentication Value
www.siebel.com	Siebel		

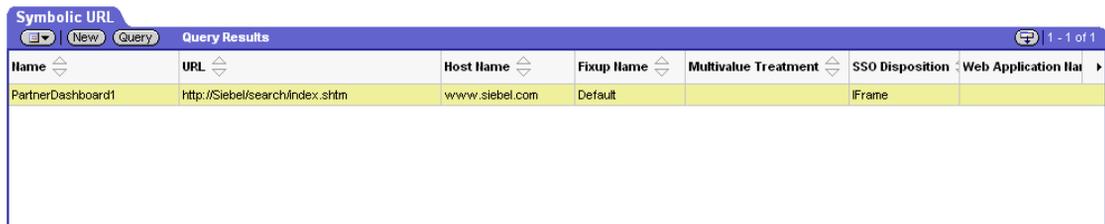
**Figure 1. External Host Administration**

For more information see [“Defining the External Host”](#) on page 30

## Define the Symbolic URL

After you define the external host you can define the symbolic URL. Notice that the URL defined here uses the Virtual Name of the host, not the actual name. Also notice that when you select the external host from the Host Name field, it is populated with the actual host name. When SWE constructs the URL, it substitutes the actual Host Name for the Virtual Name in the URL. In this example, the fixup type is Default because the page will be displayed in the browser using the `<iframe>` tag and therefore, links should not be fixed up in any way.

Figure 2 shows the Symbolic URL defined for this example.



The screenshot shows a web application window titled "Symbolic URL" with a "Query Results" tab. The table below is the result of a query.

Name	URL	Host Name	Fixup Name	Multivalue Treatment	SSO Disposition	Web Application Ha
PartnerDashboard1	http://Siebel/search/index.shtml	www.siebel.com	Default		Iframe	

**Figure 2. Symbolic URL**

For more information about defining Symbolic URLs, see [“Defining Symbolic URLs” on page 32.](#)

### Define Symbolic URL Arguments

You use symbolic URL Arguments to define the information that you want to append as arguments to the URL. You also use Symbolic URL Arguments to define commands that you want to execute. In this case, the following arguments commands are required:

- **PostRequest.** This command instructs SWE to submit the request using a POST method rather than GET, which is the default. In this case, you know POST is required because the method attribute of the form tag specifies POST.
- **UserLoginPassword.** This command instructs SWE to retrieve the password stored for the user and pass it to the external application. The name of this argument is the name of the input field expected by the external application. In this case, it is *password*.
- **UserLoginID.** This command instructs SWE to retrieve the stored login name for the user and pass it to the external application. The name of this argument is the name of the input field expected by the external application. In this case, it is *username*.

Figure 3 shows the symbolic URL arguments defined for this example.



The screenshot shows a table titled "Symbolic URL Arguments" with a header row and three data rows. The header row includes columns for Name, Required Argument, Argument Type, Argument Value, Argument Value, Append as Argun, Substitute in Text, and Sequence #. The data rows are for username, password, and anyname, each with a checkmark in the Required Argument column and a specific command in the Argument Value column.

Name	Required Argument	Argument Type	Argument Value	Argument Value	Append as Argun	Substitute in Text	Sequence #
username	✓	Command	UserLoginId	✓			1
password	✓	Command	UserLoginPassword	✓			2
anyname		Command	PostRequest				3

**Figure 3. Symbolic URL Arguments**

For more information about Symbolic URL arguments, see [“Defining Symbolic URL Arguments” on page 35.](#)

For more information about Symbolic URL commands, see [“Portal Agent Command Reference” on page 46.](#)

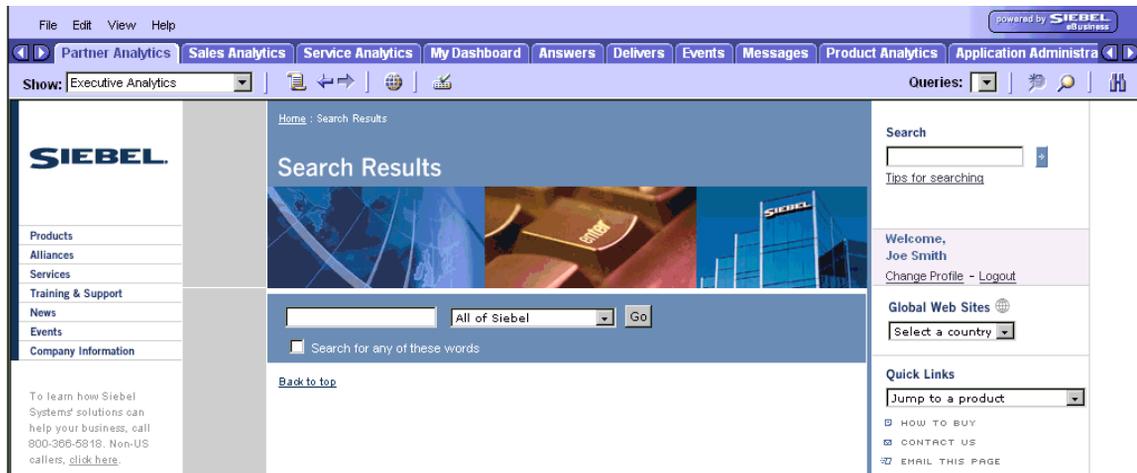
## Define User Login Credentials

Finally you must define login credentials for a user. The values defined here will be appended as arguments to the URL constructed by SWE. In this case, the following username and password are defined:

- username = Joe\_Smith@yahoo.com
- password = abracadabra

## Test

After completing the previous steps, you can test the integration. Log out of the application, and then log back in as the test user. Navigate to the applet or Web page item that is associated with the Symbolic URL. Content from the external host, in this case Siebel.COM, is displayed in the Siebel user interface, as shown in [Figure 4](#). Notice that the Joe Smith is logged into Siebel.COM.



**Figure 4. External Content Displayed in the Siebel User Interface**

# Reviewing the SWE Log File

The SWE log file can help you debug errors in your Portal Agent configuration.

- The location of the log file is, `siebsrvr_root\log`.
- The name of the log file is, `swelog_xxx.log`, where `xxx` = process id.

The log file is not enabled by default.

### To enable the SWE Log file

- 1 Open your application's configuration file, for example `uagent.cfg`.

Application configuration files are located in the `siebsrvr_root\BIN\language_code`.

- 2 Find the parameter `SWELOG` and change its value to `Enable`.

`SWELOG = Enable`

# Portal Agent Command Reference

The following commands allow you to do things such as use a set of stored credentials for authentication or define additional attributes for the `<iframe>` tag. These commands are entered as Symbolic URL Arguments. See [“Defining Symbolic URLs” on page 32](#).

## EncodeURL

- Usage** Use the `EncodeURL` command to specify whether or not the symbolic arguments should be encoded when appended to the URL. By default the URL is encoded. However, some servers do not recognize standard encoding, in which case you can use this command to not encode the URL.

**Symbolic  
URL  
Arguments**

Define the following fields in the Symbolic Arguments applet:

Field	Value
Name	EncodeURL
Argument Value	TRUE or FALSE

**IFrame****Usage**

Use the IFrame command to define additional HTML attributes for the <iframe> tag.

**Symbolic  
URL  
Arguments**

Define the following fields in the Symbolic URL Arguments applet:

Field	Value	Example
Name	Any Name	
Argument Value	IFrame [ <i>attribute</i> ] - [ <i>value</i> ]	IFrame Height = 100 Width = 500

**Disposition  
Types**

Use the IFrame disposition type with the IFrame disposition type.

**NoCache****Usage**

Use the NoCache command to instruct SWE not to cache Inline responses on the server. This command is only valid for the Inline disposition type.

**Symbolic  
URL  
Arguments**

Define the following fields on the Symbolic URL Arguments applet:

Field	Value
Name	Any name
Argument Value	NoCache

### NoFormFixup

**Usage** Use the NoFormFixup command to Instruct SWE not to fix up a form by putting proxy SWE arguments into links that appear on the page.

**Symbolic URL Arguments** Define the following fields in the Symbolic URL Arguments applet:

Field	Value
Name	Any name
Argument Value	NoFormFixup

## PreLoadURL

**Usage** Use this command to specify a preloaded URL. Use this command when the external application gathers information from a preloaded cookie on the client machine. Use this command with disposition types of IFrame and Web Control.

**Symbolic  
URL  
Arguments**

Define the following fields in the Symbolic URL Arguments applet:

Field	Value
Name	PreLoadURL
Argument Value	[URL]

## PostRequest

**Usage** Use PostRequest to configure the Portal Agent to use the POST method instead of the GET method, which is the default. Use this command when the method of the action attribute is POST. This method avoids displaying user information on a Web page or browser status bar. Use this command with disposition types of IFrame and Web Control only.

**Symbolic  
URL  
Arguments**

Define the following fields in the Symbolic URL Arguments applet:

Field	Value
Name	Any Name
Argument Value	PostRequest

### UserLoginId

**Usage** Use the UserLoginId command to send the stored user login ID for a particular Web application. The command gets the user's Login ID from the My Login Credential business component.

See [“Defining End-User Login Credentials” on page 40](#) for more information about how user login IDs are entered into this business component.

**Symbolic  
URL  
Arguments**

Define the following fields in the Symbolic URL Arguments applet:

Field	Value
Name	<i>[input field name]</i>
Argument Value	UserLoginId

### UserLoginPassword

**Usage** Use the UserLoginPassword command to send the stored user password for a particular Web application. The command gets the user's password from the My Login Credential business component.

See [“Defining End-User Login Credentials” on page 40](#) for more information about how user passwords are entered into this business component.

**Symbolic  
URL  
Arguments**

Define the following fields in the Symbolic URL Arguments applet:

Field	Value
Name	<i>[input field name]</i>
Argument Value	UserLoginPassword

## UseSiebelLoginId

**Usage** Use the UseSiebelLoginId command to retrieve the user's Siebel login ID from the stored set of credentials.

**Symbolic URL Arguments** Define the following fields in the Symbolic URL Arguments applet:

Field	Value
Name	[input field name]
Argument Value	UseSiebelLoginId

## UseSiebelLoginPassword

**Usage** Use the UseSiebelLoginPassword command to retrieve the user's Siebel password from the stored set of credentials.

**Symbolic URL Arguments** Define the following fields in the Symbolic URL Arguments applet:

Field	Value
Name	[input field name]
Argument Value	UseSiebelLoginPassword

### WebControl

**Usage** Use the WebControl command to define additional HTML attributes for Portal Agents with a disposition type of Web Control.

**Symbolic URL Arguments** Define the following fields in the Symbolic URL Arguments applet:

Field	Value	Example
Name	Any Name	
Argument Value	WebControl [ <i>attribute</i> ] - [ <i>value</i> ]	WebControl Height = 100 Width = 500

This chapter describes how to use the XML Web Interface to deliver content to external portal frameworks and Web application environments. The XML interface provides industry-standard integration to third-party development environments, such as ASP and JSP, as well as providing a model consistent with emerging Web technologies. The XML interface can be used across all Siebel eBusiness Applications, although some specialized applets may have limited support for this interface.

Developers can configure Siebel applications to support different markups, such as cHTML and xHTML, by combining the XML interface with XSL style sheets and the eAI XSLT business service.

## Overview of the XML Web Interface

The XML interface provides access to Siebel eBusiness Applications through the Siebel Web Engine (SWE). SWE generates user interface, in HTML or WML, using views, applets, and templates. These UI constructs provide access to and filtering for business object and business component data. They also provide access to visibility, navigation, and security. By rendering the XML based on the underlying SWE technology, the XML interface exposes business object and business component data, and UI elements and constructs, such as visibility, navigation, edit presence, personalization, and security. Most Siebel applets, with the exception of applets based on certain specialized applet classes, can be rendered in XML through the XML interface. The XML interface can be invoked using the following methods:

- Server configuration parameters
- Inbound URL query string parameters
- Inbound HTTP post of XML document
- Using the Web Engine Interface business service

# Accessing Siebel XML

By default, Siebel eBusiness Applications present a standard HTML-based user interface (UI) to end users. When you use the XML interface, the standard architecture changes slightly; an XML interface layer is introduced. The XML interface layer accesses Siebel eBusiness Applications through the SWE using the UI constructs, views, applets, and templates. It provides visibility into Siebel business objects and business components. These UI constructs provide not only filtering and access to business object and business component data, but also provide access to visibility, navigation, and security.

You can use the XML interface to retrieve data and UI constructs from your Siebel eBusiness Application and display it to end users according to your business needs. You can also combine this interface with XSL style sheets and the XSLT business service to generate custom HTML or other markup languages directly from the Siebel application.

For example, you can display a Siebel view using XML format rather than HTML by using a SWE command to set the markup language to XML. This example uses the Account view as an example.

### **To view the Accounts view in XML**

- 1** Log in to your Siebel application.
- 2** Type the following SWE commands and arguments appended to the URL in your browser:

```
SWEcmd=GotoPageTab&SWEScreen=Accounts+Screen&SWESetMarkup=XML
```

For example, using the mobile Web client, the URL would look like the following:

```
http://localhost/start.swe
?SWECmd=GotoPageTab&SWEScreen=Accounts+Screen&SWESetMarkup=XML
```

The Accounts view is rendered in XML format as shown in the following figure.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <APPLICATION NAME="Siebel Universal Agent">
  <USER_AGENT MARKUP="HTML" />
  - <NAVIGATION_ELEMENTS>
    - <MENU NAME="AppletMenu1">
      <MENU_ITEM TYPE="Separator" />
      - <MENU_ITEM TYPE="Command" ENABLED="TRUE" NAME="New Record">
        - <ANCHOR PATH="/start.swe">
          - <CMD VALUE="InvokeMethod" NAME="SWECmd">
            <ARG NAME="SWEApplet">Account List Applet</ARG>
            <ARG NAME="SWEView">Account List View</ARG>
            <ARG NAME="SWERowId">1-6</ARG>
            <ARG NAME="SWENeedContext">>true</ARG>
            <ARG NAME="SWEReqRowId">0</ARG>
            <ARG NAME="SWEMethod">NewRecord</ARG>
          </CMD>
          <INFO NAME="SWEC">5</INFO>
        </ANCHOR>
      </MENU_ITEM>
    </MENU>
  </NAVIGATION_ELEMENTS>
</APPLICATION>
```

## Siebel OM and Web Server Configuration and Markup Determination

The Siebel Web Engine (SWE) can be configured to produce output in HTML, WML, and XML markup languages. The default markup for a given object manager is set using the SWEMarkup parameter in the SWE section of the object manager configuration file. Based on browser or device detection or parameters set on the inbound request, this default markup may be overridden.

The following is a summary of how the markup will be determined for a given request. The following three steps are used in the markup determination process for a given request. They are listed by priority.

- 1 Inbound request, SWESetMarkup="XML, WML or HTML".** This is an optional inbound request parameter that can be used to override the User Agent Service and Server configuration. Valid values for this are XML, WML, or HTML. The User Agent Service and server configuration are not used to determine the markup when the SWESetMarkup parameter is defined on the inbound request.
- 2 User agent service.** This service is used to determine the markup based on the device or browser that generated the request. The service will take information from the request header and look up the designated markup in the device table. The resulting markup is passed to the next step. Note, if no match is found in the device table the default markup is HTML.
- 3 Dynamic markup comparison.** Assuming that no markup is specified by the inbound request SWESetMarkup parameter, the markup from the user agent service is compared to the server default configuration to determine what markup will be generated. The server default markup is designated by the SWEMarkup parameter in the OM .cfg file.

Table 6 shows a summary of the markup that will be generated for a given request based on the intersection of the server configuration markup and the markup from the user agent service.

**Table 6. Markup Summary**

Server Configuration Value	User Agent Markup Value		
	HTML	WML	XML
HTML	HTML	HTML	XML
WML	XML	WML	XML
XML	XML	XML	XML

## Accessing Specialized WML Behavior

When using the XML interface in conjunction with the Siebel Wireless WML based application the “Wireless” parameter must be set to TRUE in the Siebel Wireless OM configuration file.

---

**NOTE:** Wireless = TRUE is the default value in the configuration file when you install the Siebel Wireless application server.

---

For more information on using the XML interface with the Siebel Wireless application, please see *Siebel Wireless Administration Guide*.

## Connecting to the XML Web Interface

The XML Web Interface can be used against any Siebel eBusiness Application. Requests to generate XML from Siebel eBusiness Applications can be submitted through a Siebel Web Server or using the Web Engine Interface business service. In both cases, requests can take the form of a query string or an XML command block.

## Submitting HTTP Requests Through the Web Server

You can send HTTP requests to SWE through the Siebel Web Server using a query string or XML command block. Examples of these two methods are provided in the following sections.

### Query String

You can send HTTP requests to SWE using a query string. For example, the following code sample illustrates an Active Server Page that uses MSXML to make an HTTP request. The request logs in to the Siebel application and navigates to the Accounts screen. The XML response from SWE is transformed into HTML using XSLT.

```
<% @LANGUAGE="VBScript" %>
```

```
<%
```

```
'-----
```

```
'Open HTTP connection and send XML command req
'-----

    strURL = "http://" & Request.form ("swe") & "/"
start.swe?SWECmd=ExecuteLogin&SWEDataOnly=1&SWEUserName=sadmin&SWE
Password=sadmin&SWESetMarkup=XML
ZOSet xmlhttp = Server.CreateObject("MSXML2.ServerXMLHTTP")
xmlhttp.open "GET", strURL, False
xmlhttp.send ()
Set ologinXmlDoc = xmlhttp.responseXML

strCookie = xmlhttp.getResponseHeader ("Set-Cookie")
On Error Resume Next
If strCookie = "" Then
Response.Write ("Unable to connect to Siebel Web Server. Please
check Login Name, Password, and Siebel Web Server URL")
Response.End

End If
strSessionId =
mid(strCookie,inStr(strCookie,"!"),inStr(strCookie,";")-
inStr(strCookie,"!"))

strURL = "http://" & Request.form ("swe") & "/"
start.swe?SWECmd=GotoPageTab&SWEScreen=Accounts+Screen&SWESetMarku
p=XML&SWEDataOnly=1" & "&_sn=" & strSessionId
Set xmlhttp = Nothing
Set xmlhttp = Server.CreateObject("MSXML2.ServerXMLHTTP")
xmlhttp.open "GET", strURL, False
xmlhttp.send ()
Set oXmlDoc = xmlhttp.responseXML

'-----

'Session Var
'-----

Session ("SWESessionId") = strSessionId
Session ("swe") = Request.form ("swe")

'-----

'Prepare XSL
'-----
```

```

sXsl = "acctresponse.xsl"
Set oXslDoc = Server.CreateObject("Msxml2.DOMDocument")
oXslDoc.async = false
oXslDoc.load(Server.MapPath(sXsl))

%>

<HTML>

<HEAD>

<TITLE>My Portal</TITLE>...

<BODY>

...

<TD colSpan=2><%Response.Write (oXmlDoc.transformNode(oXslDoc))%>
</TD>

...

</BODY>

</HTML>

```

## XML Command Block

You can use an XML command block to send the HTTP request through the Siebel Web server. For example, you can submit inbound XML documents to SWE as the HTTP request body data. In the Java code sample below, the XML command block opens a socket connection to the Web server and writes the request data stream to the socket's OutputStream.

```

public static final String FULL_XML_PROC_STR = "<?xml version=\"1.0\"
encoding=\"UTF-8\"?>\n";

InputStreamin;
BufferedReaderfromServer;
PrintWriter toServer;
Socketsocket;
String payload;
Stringline;

try
{
    if (request != null && request.length() > 0)

```

```
        {  
            // send request  
            socket      = new Socket(url.getHost(), url.getPort());  
  
            toServer    = new PrintWriter(new  
OutputStreamWriter(socket.getOutputStream()));  
            in          = socket.getInputStream();  
  
            payload = FULL_XML_PROC_STR + request;  
            toServer.println("POST " + url.toString() + " HTTP/1.0");  
  
            toServer.println("Cookie: " + sessionID);  
            toServer.println("Content-Type: text/xml");  
            toServer.print("Content-Length: ");  
            toServer.println(payload.length());  
            toServer.println("");  
            toServer.println(payload);  
            toServer.flush();  
  
            fromServer = new BufferedReader(new InputStreamReader(in));  
  
            // read the response  
            while ((line = fromServer.readLine()) != null)  
            {  
                . . .  
            }  
  
            fromServer.close();  
            toServer.close();  
            socket.close();  
        }  
    }  
  
    catch (Exception ex)  
    {  
        System.err.println(ex.toString());  
    }  
}
```

## Submitting Requests Using the Web Engine Interface

Using Siebel's Object Interfaces you can access business services. The Web Engine Interface business service allows you to make requests to the Siebel Web Engine without having to submit requests through the Web server. Commands can be a query string or XML command block. The response is in the XML output property set of the business service.

For example, using the `com.siebel.data.SiebelDataBean` JavaBean class, you can access SWE as a business service from within the JSP environment. The `SiebelDataBean` provides a direct connection to SWE using the SISNAPI protocol; you can establish this connection without going through a Web server and the Siebel Web Extension plug-in. Once the data bean is instantiated, the SWE business service is obtained as an instance of `com.siebel.data.SiebelService` by calling the `getService()` method on the data bean with the service name “Web Engine Interface.”

The following code fragments show four tasks required to make the request:

- 1** Instantiate the business service.
- 2** Define the input properties.
- 3** Invoke the method.
- 4** Define the output property.

---

**NOTE:** For the complete code sample, see [“Web Engine Interface Code Sample” on page 67](#)).

---

For example, the following code fragment instantiates the Web Engine Interface business service:

```
{
    SiebelDataBean sdBean = new SiebelDataBean();
    sdBean.login(conn, uName, passwd, locale);
    SiebelService service = sdBean.getService("Web Engine
    Interface");
    . . .
}
```

The following code fragment defines the input properties for invoking the request method:

```
{
    SiebelPropertySet pi = sdBean.newPropertySet();

    pi.setProperty("J", "0"); // Mode
    pi.setProperty("E", getServerName()); // ServerName
    pi.setProperty("F", getServerPort()); // ServerPort
}
```

```
        pi.setProperty("A", "POST");           // Method
        pi.setProperty("B", "");               // RequestURI
    }
```

You can invoke SWE business service methods by calling the `SiebelService` class method `InvokeMethod()` and passing in the method name and the input and output property sets (`com.siebel.data.SiebelPropertySet`). Results from the method invocation are generated as a property in the output property set. Here is a sample code fragment for invoking the “Start” method:

```
{

    SiebelPropertySet pi = sdBean.newPropertySet();
    SiebelPropertySet po = sdBean.newPropertySet();

    printInfo("Initializing SWE.....");
    pi.setProperty("A", getUsername());        // UserName
    pi.setProperty("B", "80");                 // HTTPPort
    pi.setProperty("C", "443");                // HTTPSPort
    pi.setProperty("D", getServerName());     // ServerName
    pi.setProperty("G", "F");                  // UseCookie
    // initialize the swe
    if (swe.invokeMethod("Start",pi,po))
    {
        printInfo("Done initializing SWE!");
        //      printPS(pi, "Input PropertySet");
        printPS(po, "Login Output PropertySet");
    }
    else
    {
        printInfo("Failed to initialize SWE!");
    }
}
```

### Query String

Using the Web Engine Interface, you can send requests to SWE either as a URL query string or as an XML document in the request body data. To send a request using a URL, set the property “QueryString” to the URL query string with all the SWE commands and parameters.

For example, the following code sample defines the input properties (Step 2 in the SiebelDataBean example) using a query string:

```
{
    SiebelPropertySet pi = getInputPropertySet(query);
    String query =
    "SWECmd=ExecuteLogin&SWEDataOnly=1&SWEUserName=SADMIN&SWEPassword=
    SADMIN&SWESetMarkup=XML";
    pi.setProperty("C", query);           // QueryString
}
```

The following code fragment invokes the Request Web Engine Interface business service method (Step 3 in the SiebelDataBean example):

```
boolean result = swe.invokeMethod("Request", pi, po);
```

---

**NOTE:** The Request method is the most commonly used Web Engine Interface method.

---

## XML Command Block

To send a request using an XML command block, set the input properties “RequestBodyData,” “RequestBodyLength,” and “RequestBodyLengthTotal.” The property “RequestBodyData” must contain the XML request as specified in “Outbound DTD” on page 120. You should either leave the query string blank or make sure it does not contain the parameter “SWECmd” or its short form “C” (for example, SWECmd = GotoView or C = Gv).

For example, the following code sample defines the input properties (Step 2 in the SiebelDataBean example) using an XML command block:

```
{
    String xmlRequest = "<EXEC>" +
    "<CMD NAME=\"SWECmd\" VALUE=\"GotoPageTab\">" +
    "<ARG NAME=\"SWEScreen\">Opportunities Screen</ARG>" +
    "<ARG NAME=\"SWESetMarkup\">XML</ARG>" +
    "</CMD>" +
    "</EXEC>";

    xmlRequest = "<?xml version=\"1.0\"?>" + xmlRequest;
    String len = (new Integer(xmlRequest.length() *
    2)).toString();
    pi.setProperty("S", len);           // RequestBodyLength
}
```

```
        pi.setProperty("T", len);          // RequestBodyLengthTotal
        pi.setProperty("U", xmlRequest);   // RequestBodyData
    }
```

The following code fragment invokes the Request Web Engine Interface business service method ([Step 3](#) in the SiebelDataBean example):

```
boolean result = swe.invokeMethod("Request", pi, po);
```

## Web Engine Interface Property Names

This section describes the property names for the Web Engine Interface business service.

### Start Method Properties

[Table 7](#) lists the property names for the Start Method.

**Table 7. Property Names for Start Method**

Property Name	Description
A	User name
B	HTTP port
C	HTTPS port
D	Server name
E	Session ID
F	Cookie name
G	Use cookie
H	Sticky
I	Single sign on (SSO)
J	External service name
K	Swe plugin type

## Request Method Properties

Table 8 lists the property names for the Request Method.

**Table 8. Property Names for the Request Method**

Property Name	Description
A	Method
B	Request URI
C	Query string
D	Channel
E	Server name
F	Server port
G	Script name
H	Client
I	Use cookie
J	Mode
K	Remote user
L	From timeout session
M	Cb stop
N	Cookie
O	Request headers
P	Modification
Q	Request body type
R	Request more body
S	Request body length
T	Request body length total
U	Request body data

**Table 8. Property Names for the Request Method**

Property Name	Description
V	Request source (if from plug-in)
W	Request source (if guest session)

### Response Properties

[Table 9](#) lists the Response properties.

**Table 9. Response Properties**

Property Name	Description
A	Call back
B	Result code
C	Set cookies
D	Response headers
E	Response type
F	Response Length
G	Response data
ERR	Error key
MSG	Error message
CP	Code page
XHTML	XHTML
ENC	Encoding (base64, ...)

## Web Engine Interface Code Sample

The following code sample demonstrates invoking the Web Engine Interface business service.

```

////////////////////////////////////
//
// Copyright (C) 2003 Siebel Systems, Inc.,
// All rights reserved.
//
// FILE:      SWEBean.java
//   $Revision: $
//   $Date: $
//   $Author:$ of last update
//
// CREATOR:
//
// DESCRIPTION
//   This class encapsulates the initialization of SiebelDataBean and SWE
//   Web Service, and simplifies the communications between SWE and JSPs
//
////////////////////////////////////

import com.siebel.data.*;
import java.io.BufferedReader;
import java.io.InputStreamReader;

//-----
public class SWEBean
{
// SWE service name
static final String      SWE_SERVICE_NAME = "Web Engine Interface";

// instance variables
private SiebelDataBean  sdBean           = null;
private SiebelService   swe              = null;
private String          userName         = null;
private String          password         = null;
private String          connectString    = null;
private String          serverPort       = null;
private String          serverName       = null;

private String          locale           = "enu";
//-----
public SWEBean()
{
}
}

```

```
//-----  
// Entry point for testing  
public static void main(String[] args)  
{  
    String testCmd = "<EXEC>" +  
        "<CMD NAME=\"SWECmd\" VALUE=\"GotoPageTab\">" +  
        "<ARG NAME=\"SWEScreen\">Opportunities Screen</ARG>" +  
        "<ARG NAME=\"SWESetMarkup\">XML</ARG>" +  
        "</CMD>" +  
        "</EXEC>";  
  
    try  
    {  
        String result;  
        for(int i =0; i < args.length ; i++)  
        {  
            System.out.println("Argument [" + i + "] : " + args[i]);  
        }  
        SWEBean sBean = new SWEBean();  
        sBean.setConnectString(args[0]); // connect string is something  
like "siebel://KHLEE2:2320/siebel/SCCObjMgr_enu/KHLEE2"  
        sBean.setUsername(args[1]); // testuser  
        sBean.setPassword(args[2]); // db2  
        sBean.setServerName(args[3]); // khlee2  
        sBean.setServerPort(args[4]); // 2320  
        if (args.length > 5)  
        {  
            sBean.setLocale(args[5]);  
        }  
  
        // connect to OM  
        sBean.init();  
  
        //String testQ =  
"SWECmd=InvokeMethod&SWERegrowId=1&SWEField=s_1_2_2_1&SWEMethod=Drilldown&SWERowI  
d0=1-959&SWEView=Product+List+View&SWEApplet=Product+List+Applet&SWERowId=1-  
959&SWENeedContext=true&SWESetMarkup=XML";  
        //result = sBean.makeXMLRequest("/callcenter_enu/start.swe",  
sBean.getUserName(), false, testQ);  
        result = sBean.makeXMLRequest(testCmd, sBean.getUserName(), false,  
"");  
  
        if (result != null)  
        {  
            System.out.println("Result : " + result);  
        }  
        else  
        {  
            System.out.println("Result : null");  
        }  
    }  
    catch (Throwable t)
```

```

        {
            t.printStackTrace(System.err);
        }
    }
}
//-----
public void init() throws Exception {
    init(userName, password, connectString);
}

//-----
protected void init(String uName, String passwd, String conn) throws Exception
{
    if (uName != null && passwd != null && conn != null)
    {
        SiebelDataBean tBean = new SiebelDataBean();
        printInfo("Logging in to SiebelDataBean.....");
        printInfo(" tBean.login(" + conn + ", " + uName + ",\n " +
            " " + passwd + ", " + getLocale() + ")");
        tBean.login(conn, uName, passwd, getLocale());
        printInfo("Finished login!");
        sdBean = tBean;

        initSWE();
    }
    else
    {
        throw new Exception("Invalid login parameters!");
    }
}

//-----
protected void initSWE() throws SiebelException
{
    printInfo("Getting SWE Service.....");
    swe = getService(SWE_SERVICE_NAME);
    printInfo("Finished getting SWE Service!");

    //pause();

    SiebelPropertySet pi = sdBean.newPropertySet();
    SiebelPropertySet po = sdBean.newPropertySet();

    /**
    the important property here is username because it tells
    the SWE that the user has logged in, which is different
    than telling swe who the user is, which we can do by setting
    the UserName in the args for a normal Request.
    */
    printInfo("Initializing SWE.....");
    pi.setProperty("A", getUser_name()); // UserName
}

```

```
pi.setProperty("B", "80"); // HTTPPort
pi.setProperty("C", "443"); // HTTPSPort
pi.setProperty("D", getServerName()); // ServerName
pi.setProperty("G", "F"); // UseCookie
// initialize the swe
if (swe.invokeMethod("Start",pi,po))
{
    printInfo("Done initializing SWE!");
    // printPS(pi, "Input PropertySet");
    printPS(po, "Login Output PropertySet");
}
else
{
    printInfo("Failed to initialize SWE!");
}
}

//-----
protected SiebelService getService(String sName) throws SiebelException
{
    if (sName != null && sdBean != null)
    {
        SiebelService service = sdBean.getService(sName);
        return service;
    }
    return null;
}

//-----
public String makeXMLRequest(String xmlRequest,
                             String remoteUser,
                             boolean isSecure,
                             String query) throws SiebelException
{
    printInfo("Processing XMLRequest : " + xmlRequest);
    printInfo("Getting ExecPath.....");
    //String execPath = getExecPath(xmlRequest);
    //printInfo("Done getting ExecPath : " + execPath);

    SiebelPropertySet pi = sdBean.newPropertySet();
    SiebelPropertySet po = sdBean.newPropertySet();

    pi.setProperty("J", "0");// Mode
    pi.setProperty("E", getServerName()); // ServerName
    pi.setProperty("F", getServerPort()); // ServerPort
    pi.setProperty("A", "POST"); // Method
    pi.setProperty("B", ""); // RequestURI
    pi.setProperty("Q", "text/xml"); // RequestBodyType
}
```

```

if (xmlRequest.equals(""))
{
    pi.setProperty("S", "0");           // RequestBodyLength
    pi.setProperty("T", "0");           // RequestBodyLengthTotal
    pi.setProperty("U", "");            // RequestBodyData
}
else
{
    xmlRequest = "<?xml version=\\"1.0\\"?> " + xmlRequest;
    String len = (new Integer(xmlRequest.length() * 2)).toString();
    pi.setProperty("S", len);           // RequestBodyLength
    pi.setProperty("T", len);           // RequestBodyLengthTotal
    pi.setProperty("U", xmlRequest);    // RequestBodyData
}
// -----
pi.setProperty("C", query);             // QueryString
pi.setProperty("G", "");                // ScriptName
pi.setProperty("D", (isSecure ? "https" : "http")); // Channel
pi.setProperty("I", "F");               // UseCookie
pi.setProperty("K", getUsername());     // RemoteUser
// -----
printPS(pi, "XMLRequest Input Property");
printInfo("Making XMLRequest.....");

boolean result = swe.invokeMethod("Request", pi, po);

if (result)
{
    printInfo("Done XMLRequest!");
}
else
{
    printInfo("XMLRequest failed!");
}
printPS(po, "XMLRequest output");
String response = po.getProperty("RespData");
return response;
}

//-----
// Accessor functions
public void setUsername(String name)
{
    userName = name;
}

//-----
public String getUsername()
{
    return userName == null ? "" : userName;
}

```

```
//-----  
public void setPassword(String passwd)  
{  
    password = passwd;  
}  
  
//-----  
public String getPassword()  
{  
    return password == null ? "" : password;  
}  
  
//-----  
public void setConnectionString(String conn)  
{  
    connectionString = conn;  
}  
  
//-----  
public String getConnectionString()  
{  
    return connectionString == null ? "" : connectionString;  
}  
  
//-----  
public void setServerPort(String port)  
{  
    serverPort = port;  
}  
  
//-----  
public String getServerPort()  
{  
    return serverPort == null ? "" : serverPort;  
}  
  
//-----  
public void setServerName(String sName)  
{  
    serverName = sName;  
}  
  
//-----  
public String getServerName()  
{  
    return serverName == null ? "" : serverName;  
}
```

```

//-----
public void setLocale(String lang)
{
    locale = lang;
}

//-----
public String getLocale()
{
    return locale == null ? "enu" : locale;
}

//-----
public void pause()
{
    try
    {
        System.out.println("The batch command reader has paused.");
        System.out.print("Press Enter to resume.");
        BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
        String tmp = in.readLine();
    }
    catch (Exception ex)
    {
    }
}

//-----
protected void printInfo(String msg)
{
    System.out.println(msg);
}

//-----
void printPS(SiebelPropertySet ps, String title)
{
    // Testing
    printInfo("***** " + title + " *****");
    for (String name = ps.getFirstProperty(); name != null &&
!name.equals(""); name = ps.getNextProperty())
    {
        String value = ps.getProperty(name);
        printInfo(name + " = " + value);
    }
}
}

```

# XML Request Structure

The XML API offers developers access to the objects within Siebel eBusiness Applications. Although it is not required that you have a complete understanding of Siebel object definitions and architecture, it is strongly recommended that you be familiar with them.

You can structure requests using a query string or a command block.

## Query String

To construct a request using a query string, you append SWE commands and arguments to a URL. Each command or argument and its value is separated by an “&”. For example:

```
SWECmd=ExecuteLogin&SWEDataOnly=1&SWEUserName=sadmin&SWEPassword=sadmin&SWESetMarkup=XML
```

For a list of commonly used SWE commands and arguments see [“SWE API” on page 104](#).

## XML Command Block

To initiate an action on a Siebel eBusiness XML screen, you must use a specific set of XML tags and they must conform to a specific structure. [Table 10](#) lists the three valid XML tags that are used to perform a command.

**Table 10. XML Tags**

Tag	Description
< EXEC >	The root tag for each command that you want to send to the SWE. The < EXEC > tag encloses the < CMD > and < ARG > tags. This tag represents a single command.

**Table 10. XML Tags**

Tag	Description
<CMD>	This tag indicates the SWE command that you want to access and encloses all arguments for the command.
<ARG>	This tag indicates the object on which the command is to be executed and any additional parameters that are required. Unlike the <EXEC> and <CMD> tags, which are used only once in a command block, you can have multiple arguments within a command block.

For example, using the information from [Table 10](#), a valid syntax format for an XML command block is as follows:

```
<EXEC>
<CMD NAME="SWEcmd" VALUE="command name">
<ARG NAME="argumentName">argument1Value</ARG>
<ARG NAME="argumentName">argument2Value</ARG>
...
<ARG NAME="argumentName">argumentNValue</ARG>
<ARG NAME="SWESetMarkup"> XML | HTML </ARG>
<ARG NAME="SWEDataOnly"> TRUE | FALSE </ARG>
<ARG NAME="SWESetNoTempl"> TRUE </ARG>
</CMD>
</EXEC>
```

Each <EXEC> tag encloses a complete command block. The <CMD> and <ARG> tags are enclosed within the <EXEC> tag, and their attributes and values specify which commands are executed by the SWE.

A valid XML command block must conform to a specific structure. It must have a valid execute tag followed by a command tag that encloses the arguments. The syntax of the name-value pairs and the attributes that accompany the XML tags within a command block must follow a specific format. This section details the syntax of each XML tag. For the DTD for the inbound XML document, see [“Inbound DTD” on page 120](#).

### EXE Tag

The Execute tag is the root tag for each command that you want to execute.

### Description

Think of the Execute tag as a container. Each container represents a single SWE command or screen action. Enclosed within an Execute tag are the commands, arguments, and information required to complete a single command. There should be only one <EXEC> tag for each command that you want to execute. The PATH attribute is the only attribute used by the <EXEC> tag, although it is not required.

### Attributes

[Table 11](#) lists the attributes used with the Execute tag:

**Table 11. EXEC Tag Attributes**

Attribute	Description
PATH	The PATH attribute is used to indicate the location of the SWE object manager. By default, the eBusinessSWEXML application looks in its root directory for the SWE object manager. If you want to specify an object manager for the Web application to use, you must indicate its location using the PATH attribute.

### Example

The following example uses the Execute tag to enclose the login command.

```
<?xml version="1.0" encoding="UTF-8"?>
<EXEC PATH="/callcenter/start.swe">
  <CMD NAME="SWECmd" VALUE="ExecuteLogin">
    <ARG NAME="SWEUserName">jdoe</ARG>
    <ARG NAME="SWEPassword">jdoepassword</ARG>
    <ARG NAME="SWESetMarkup">XML</ARG>
    <ARG NAME="SWEDataOnly">TRUE</ARG>
    <ARG NAME="SWESetNoTempl">TRUE</ARG>
  </CMD>
</EXEC>
```

## CMD Tag

The Command tag is required for each command block and is used to indicate the SWE command that you want to execute.

### Description

Like the Execute tag, the Command tag also acts as a container. Enclosed between the open and close Command tags are the arguments required to complete a command. There should be only one <CMD> tag for each command block that you want to execute.

### Attributes

Table 12 lists the attributes that are used with the Command tag:

**Table 12. CMD Tag Attributes**

Attribute	Description
NAME	The NAME attribute should always be set to "SWECmd". This indicates that the type of command you want to execute is a SWE command.
VALUE	The VALUE attribute specifies which SWECmd you want to execute. Listed below are the SWE commands most commonly used with eBusiness: <ul style="list-style-type: none"> <li>■ ExecuteLogin</li> <li>■ GotoPageTab</li> <li>■ InvokeMethod</li> <li>■ LogOff</li> </ul>

### Example

Using the information from the table above, the following example illustrates how to use the Command tag to execute a login command:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXEC PATH="/callcenter/start.swe">
<CMD NAME="SWECmd" VALUE="ExecuteLogin">
<ARG NAME="SWEUserName">jdoe</ARG>
<ARG NAME="SWEPassword">jdoepassword</ARG>
```

```
<ARG NAME="SWESetMarkup">XML</ARG>
<ARG NAME="SWEDataOnly">TRUE</ARG>
<ARG NAME="SWESetNoTemp1">TRUE</ARG>
</CMD>
</EXEC>
```

### **ARG Tag**

A command block can contain multiple Argument tags. Each Argument tag indicates an additional command parameter required to complete the action specified in the command block.

### **Description**

The Argument tag uses name/value pairs to send command parameters to the SWE. A command will not execute without having all the appropriate parameters passed to the SWE.

## Attributes

[Table 13](#) lists the attributes that are used with the Argument tag.

**Table 13. ARG Tag Attributes**

Attribute	Description
NAME	<p>This is the only attribute used by the Argument tag. The NAME attribute is used to indicate an argument, or the name of a parameter, for which you are sending additional information. The parameter's value is entered between the open and close Argument tags.</p> <p>Listed below are the parameter names most commonly used with eBusiness:</p> <ul style="list-style-type: none"> <li>■ SWEApplet</li> <li>■ SWEDataOnly</li> <li>■ SWEMethod</li> <li>■ SWEPassword</li> <li>■ SWEScreen</li> <li>■ SWESetNoTempl</li> <li>■ SWESetMarkup</li> <li>■ SWESetRowCount</li> <li>■ SWEShStyleSheet</li> <li>■ SWEUserName</li> <li>■ SWEView</li> </ul> <p><a href="#">Table 14</a> lists the values that are most commonly used with these parameter names.</p>

#### Example

For each argument name that you include in a command block, you must also indicate a value for the argument. For example, to use the InvokeMethod command, you must indicate which method you want to invoke. Additionally, if the method is one that requires parameters, as is the case with the WriteRecord, you must send those parameters to the SWE. With the WriteRecord method, you need to indicate the view and the applet you are working with. You also need to indicate the column to which you want to write the record, and finally you need to indicate what information you want to write. The following example illustrates how to use Argument tags to send the required parameters for a WriteRecord method:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXEC PATH="/callcenter/start.swe">
  <CMD NAME="SWECmd" VALUE="InvokeMethod">
    <ARG NAME="SWEMethod">WriteRecord</ARG>
    <ARG NAME="SWEView">Account List View</ARG>
    <ARG NAME="SWEApplet">Account List Applet</ARG>
    <ARG NAME="Lot Name">65 metal car</ARG>
    <ARG NAME="Starting Price">3.00</ARG>
    <ARG NAME="SWESetMarkup">XML</ARG>
    <ARG NAME="SWEDataOnly">TRUE</ARG>
    <ARG NAME="SWESetNoTemp">TRUE</ARG>
  </CMD>
</EXEC>hr
```

#### Required Arguments

The following three arguments are required for each command block sent to the SWE:

```
<ARG NAME="SWESetMarkup">XML | HTML | WML</ARG>
<ARG NAME="SWEDataOnly">TRUE | FALSE</ARG>
<ARG NAME="SWESetNoTemp">TRUE</ARG>
```

- **SWESetMarkup.** The SWE returns a response for each command block it receives. You can use the SWESetMarkup attribute to indicate whether a response is returned as XML, HTML, or WML.

You can also set the response markup format by allowing the User Agent (UA) service to retrieve the default markup from the UA device table, or by setting the SWESetMarkup property in the appropriate Siebel Server configuration file. The SWESetMarkup tag is not required in the payload when you use one of these alternatives.

---

**NOTE:** The examples in this chapter specify the response markup format using the SWESetMarkup attribute in the payload.

---

- **SWEDataOnly.** In addition to specifying the type of markup language for a SWE response, you must also indicate whether the response should include data only or data and user interface information, such as non-data controls (anchors and navigation controls). You can set the SWEDataOnly attribute to TRUE to indicate that only data should be returned, or you can set it to FALSE to indicate that both data and user interface information should be returned.

---

**NOTE:** If the SWEDataOnly parameter is not included, the default is FALSE.

---

- **SWESetNoTempl.** By default, Siebel eBusiness XML uses a server-side Web template to filter specific items and controls from SWE responses. When using XML, you can control whether a response will return all the information related to the request or a subset of it dictated by the Web template. Setting the attribute to TRUE makes sure that the Web template is not used and that the SWE response contains all the necessary information to complete an action. When a SWESetNoTempl attribute is set to FALSE, the Web template is used and the page items and controls specified in the template are filtered from the response.

---

**NOTE:** If the SWESetNoTempl parameter is not included, the default is FALSE.

---

### Common Name-Value Pairs

Table 14 lists commonly used argument name-value pairs.

**Table 14. ARG Parameter Name-Values Pairs**

Parameter Name	Parameter Values
SWEApplet	<i>Applet name</i>
SWEDataOnly	TRUE FALSE
SWEMethod	<ul style="list-style-type: none"><li>■ DeleteRecord</li><li>■ EditRecord</li><li>■ ExecuteQuery</li><li>■ GoToNextSet</li><li>■ GotoPageTab</li><li>■ NewRecord</li><li>■ NewQuery</li><li>■ WriteRecord</li></ul>
SWEPASSWORD	<i>Password</i>
SWEScreen	<i>Screen name</i>
SWESetMarkup	HTML XML
SWEUserName	<i>User name</i>
SWEView	<i>View name</i>

**NOTE:** When determining what arguments to define, it is a good idea to look at the XML Response. The response will include what arguments are expected.

## XML Response Structure

When you send a command block to a Siebel eBusiness SWEXML application, you access the Siebel eBusiness XML application screens. If the action specified in the command block is successfully executed, the data and all of the objects from the resulting screen are returned within an HTTP response. The format of the response is XML, HTML, or WML, depending on the SWESetMarkup setting that was sent in the request payload.

You must develop the mechanism by which your Web server handles XML responses. Using the information provided in this section you can develop a parser, a Web application, or another control to extract the necessary data from XML responses and display the appropriate information to users. For the DTD for the outbound XML document, see [“Outbound DTD” on page 120](#).

## Errors

If a command block contains an error or is unsuccessful, the specified action is not executed. Instead, the Siebel eBusiness XML user interface retains its current state and the SWE returns an error. Based on the markup format you have specified, an error response is returned as XML, HTML, or WML.

An XML error response contains an `<ERROR>` tag within the payload. Descriptive text for the error is enclosed between the open and close `<ERROR>` tags.

## XML Response

When the SWESetMarkup attribute in a command block is set to XML, the response payload from the Siebel eBusiness XML Web server is returned in XML format. The payload consists of an XML declaration followed by the core XML tags that contain and describe the data.

Each XML tag represents an object from a Siebel eBusiness XML application screen that you requested. The attributes within each tag are read-only and represent the properties of the object.

Table 15 lists the major XML tags that are returned in a response in which the SWEDataOnly attribute is set to TRUE.

**NOTE:** The response tags described in this appendix are a subset of the tags that can be returned by the SWE.

**Table 15. XML Response Tags**

Tag	Description and Attributes
< APPLICATION >	<p>The root tag for each response that is returned from the SWE. The &lt; APPLICATION &gt; tag encloses all the XML response data.</p> <p><b>Attribute:</b></p> <ul style="list-style-type: none"><li>■ NAME. This attribute indicates the name of the application from which the response is generated. For XML requests, the application name in the response will always be "Siebel eBusiness XML."</li></ul>
< SCREEN >	<p>This tag identifies the eBusiness screen that is the result of, or is accessed by, the command in your request. The &lt; SCREEN &gt; tag also encloses all of the XML tags that identify the data within the screen.</p> <p><b>Attributes:</b></p> <ul style="list-style-type: none"><li>■ CAPTION. This attribute indicates the caption of the screen.</li><li>■ ACTIVE. A value of TRUE indicates that the screen is active. A value of FALSE indicates that the screen is inactive.</li><li>■ NAME. This attribute indicates the screen name, which is used to identify the screen.</li></ul>
< VIEW >	<p>This tag identifies the eBusiness view that is the result of, or is accessed by, the command block in your request. This tag also encloses all of the XML tags that identify the data within the view.</p> <p><b>Attributes:</b></p> <ul style="list-style-type: none"><li>■ TITLE. This attribute indicates the title of the view.</li><li>■ ACTIVE. A value of TRUE indicates that the view is active. A value of FALSE indicates that the view is inactive.</li><li>■ NAME. This attribute indicates the view name, which is used to identify the view.</li></ul>

Table 15. XML Response Tags

Tag	Description and Attributes
< APPLET >	<p>This tag identifies the eBusiness applet that is the result of, or is accessed by, the command block in your request. It also encloses all of the XML tags that identify the data within the applet.</p> <p><b>Attributes:</b></p> <ul style="list-style-type: none"> <li>■ ROW_COUNTER. This attribute indicates how many records out of the entire set of records are currently displayed. The ROW_COUNTER attribute is a string of the form, <i>1 - n of N</i>.</li> <li>■ NO_DELETE. A value of TRUE indicates that the records in the applet cannot be deleted. A value of FALSE indicates that the records in the applet can be deleted.</li> <li>■ NO_EXEC_QUERY. A value of TRUE indicates that a query cannot be executed in the applet. A value of FALSE indicates that a query can be executed in the applet.</li> <li>■ NO_UPDATE. A value of TRUE indicates that the records in the applet cannot be updated. A value of FALSE indicates that the records in the applet can be updated.</li> <li>■ MODE. Indicates the mode of the applet, which can be one of the following: Base, Edit, New, Query, Sort.</li> </ul>
< APPLET > (Continued)	<p><b>Attributes:</b></p> <ul style="list-style-type: none"> <li>■ TITLE. This attribute title of the applet.</li> <li>■ NO_INSERT. A value of TRUE indicates that records cannot be inserted into the applet.</li> <li>■ CLASS. Indicates the class being used by the applet.</li> <li>■ NO_MERGE. A value of TRUE indicates that records in the applet have not been merged. A value of FALSE indicates that the records in the applet have been merged.</li> <li>■ ACTIVE. A value of TRUE indicates that the applet is active. A value of FALSE indicates that the applet is inactive.</li> <li>■ ID. This attribute indicates the applet ID, and can be used to identify the applet.</li> <li>■ NAME. This attribute indicates the applet name, which is used to identify the applet.</li> </ul>
< LIST >	<p>This tag encloses the table of records that is returned from your request. The following two tags and their subordinate tags are enclosed within the &lt; LIST &gt; tag:</p> <p>&lt; RS_HEADER &gt;</p> <p>&lt; RS_DATA &gt;</p> <p>There are no attributes associated with the &lt; LIST &gt; tag.</p>

**Table 15. XML Response Tags**

Tag	Description and Attributes
< RS_HEADER >	This tag encloses all the header information about the columns in a list that your request returns. The < COLUMN >, < METHOD >, and < ERROR > tags can be enclosed within this tag.
< COLUMN >	<p>A response can return multiple &lt; COLUMN &gt; tags. Each &lt; COLUMN &gt; tag within an &lt; RS_HEADER &gt; tag indicates another column within the parent list.</p> <p><b>Attributes:</b></p> <ul style="list-style-type: none"><li>■ NUMBER_BASED. A value of TRUE indicates that the data in the column are numeric. A value of FALSE indicates that the data are not numeric.</li><li>■ CALCULATED. A value of TRUE indicates that the data in the column are calculated from other values, as opposed to being input. A value of FALSE indicates that the data are not calculated.</li><li>■ LIST_EDITABLE. A value of TRUE indicates that the data in the column are editable. A value of FALSE indicates the data are not editable.</li><li>■ HTML_TYPE. This attribute is used to indicate the type of object that is represented in the column.</li><li>■ SCALE. A value of TRUE indicates that the data in the column are scaled. A value of FALSE indicates that the data are not scaled.</li><li>■ FIELD. This attribute indicates the field name associated with the column. The value in the field name is the same as the column name.</li><li>■ HIDDEN. A value of TRUE indicates that the data in the column are hidden on the eBusiness screen. A value of FALSE indicates that the data are visible on the eBusiness screen.</li></ul>

**Table 15. XML Response Tags**

Tag	Description and Attributes
< COLUMN >	<ul style="list-style-type: none"> <li>■ DATATYPE. This attribute indicates the Siebel data-type of the data in the column.</li> <li>■ DISPLAY_NAME. This attribute indicates the text string that would appear in the user interface.</li> <li>■ TEXT_LENGTH. This attribute indicates the maximum length of field entries in the column.</li> <li>■ TYPE. This attribute is used to indicate the type of object that is represented in the column.</li> <li>■ ID. This attribute indicates the unique ID of the column.</li> <li>■ TEXT_BASED. A value of TRUE indicates that the data in the column is text based. A value of FALSE indicates that the data is not text based.</li> <li>■ NAME. A value of TRUE indicates that the data in the column are hidden on the eBusiness screen. A value of FALSE indicates that the data are visible on the eBusiness screen.</li> <li>■ REQUIRED. A value of TRUE indicates that the data in the column are required. A value of FALSE indicates that the data are not required.</li> <li>■ READ_ONLY. A value of TRUE indicates that the data in the column are read-only and cannot be modified. A value of FALSE indicates that the data are editable.</li> </ul>
< RS_DATA >	This tag encloses table rows that are returned from your request. The < RS_DATA > tag encloses the < ROW > tag and the < ROW > tag's subordinate tags.

**Table 15. XML Response Tags**

Tag	Description and Attributes
< ROW >	<p>A response can return multiple &lt; ROW &gt; tags. Each &lt; ROW &gt; tag within an &lt; RS_DATA &gt; tag indicates another record within the table. The &lt; ROW &gt; tag encloses the &lt; FIELD &gt; tag.</p> <p><b>Attributes:</b></p> <ul style="list-style-type: none"> <li>■ <b>SELECTED.</b> This attribute indicates whether the current row is selected. A value of TRUE indicates that the row is selected. A value of FALSE indicates it is not.</li> <li>■ <b>ROWID.</b> This attribute is used to identify the row.</li> </ul>
< FIELD >	<p>A response can return multiple &lt; FIELD &gt; tags. Each &lt; FIELD &gt; tag within a &lt; ROW &gt; tag indicates another item of data within the record. The field's value is entered between the open and close &lt; FIELD &gt; tags.</p> <p><b>Attributes:</b></p> <ul style="list-style-type: none"> <li>■ <b>VARIABLE.</b> This attribute indicates the column to which the field is associated. The value of the VARIABLE attribute should coincide with the NAME attribute of a column.</li> <li>■ <b>NAME.</b> This attribute is used to identify the field. In most cases, the field name is identical to the column name.</li> </ul>

A valid syntax format for an XML response is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<APPLICATION NAME="Siebel eAuction XML">
<SCREEN CAPTION="caption" ACTIVE="TRUE" NAME="screen name">
<VIEW TITLE="title" ACTIVE="TRUE | FALSE" NAME="view name">
<APPLET ROW_COUNTER="n - N of X" NO_DELETE="TRUE | FALSE" NO_EXEC_QUERY="TRUE |
FALSE" NO_UPDATE="TRUE | FALSE" MODE="Base" TITLE="applet title" NO_INSERT="TRUE |
FALSE" CLASS="CSSSWEFrameLotList" NO_MERGE="TRUE | FALSE" ACTIVE="TRUE | FALSE"
ID="N" NAME="applet name">
<LIST>
<RS_HEADER>
<COLUMN NUMBER_BASED="TRUE | FALSE" CALCULATED="TRUE | FALSE" LIST_EDITABLE="Y |
N" HTML_TYPE="Field" SCALE="TRUE | FALSE" FIELD="Accept Less" HIDDEN="TRUE | FALSE"
DATATYPE="text" TEXT_LENGTH="255" TYPE="Field" TOTAL_REQUIRED="TRUE | FALSE"
ID="N" TEXT_BASED="TRUE | FALSE" NAME="Accept Less" REQUIRED="TRUE | FALSE"
READ_ONLY="TRUE | FALSE"/>
```

```
</RS_HEADER>

<RS_DATA>

<ROW SELECTED="TRUE | FALSE" ROWID="id number1">

<FIELD VARIABLE="column name" NAME="field name1">
field value1
</FIELD>

...

<FIELD VARIABLE="column name" NAME="field nameN">
field valueN
</FIELD>

</ROW>

...

<ROW SELECTED="TRUE | FALSE" ROWID="id number1">

<FIELD VARIABLE="column name" NAME="field name1">
field value1
</FIELD>

...

<FIELD VARIABLE="column name" NAME="field nameN">
field valueN
</FIELD>

</ROW>

</RS_DATA>

</LIST>

</APPLET>

</VIEW>

</SCREEN></APPLICATION>
```

### HTML Response

When the SWESetMarkup attribute in a command block is set to HTML, the response payload from the Siebel eBusiness Application Web server is going to be in HTML format. The HTML option allows you to display the returned data in a read-only mode. The HTML response includes all the data and navigation controls that are exposed in the user interface.

### WML Response

When the SWESetMarkup attribute in a command block is set to WML, the response payload from the Siebel eBusiness Application XML Web server is going to be in WML format.

## Common Operations

There are various combinations of XML commands you can use to execute an action in a Siebel eBusiness XML application. Each section below offers one solution for executing a Siebel eBusiness Application action.

---

**TIP:** To get a better understanding of the objects available on a specific screen, you can use a Web browser to access the user interface by navigating to the following URL: `http://<machine name>/callcenter/start.swe.<machine name>`. This is the Web server where the Siebel eBusiness Application is installed.

---

### Logging In

Logging in is required to start a new Siebel eBusiness XML session. The first command block of a new session should always be an ExecuteLogin command.

Detailed below is an example of how to construct a login command block for XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXEC PATH="/callcenter/start.swe">
<CMD NAME="SWECmd" VALUE="ExecuteLogin">
<ARG NAME="SWEUserName">user name</ARG>
```

```
<ARG NAME="SWEPassword">user's password</ARG>  
<ARG NAME="SWESetMarkup">XML</ARG>  
<ARG NAME="SWEDataOnly">TRUE</ARG>  
<ARG NAME="SWESetNoTemp">TRUE</ARG>  
</CMD>  
</EXEC>
```

### Logging Off

The last command block of a session should always be a Logoff command.

Detailed below is an example of how to construct a logoff command block for XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXEC PATH="/callcenter/start.swe">
  <CMD NAME="SWECmd" VALUE="Logoff">
    <ARG NAME="SWEUserName">user name</ARG>
    <ARG NAME="SWESetMarkup">XML</ARG>
    <ARG NAME="SWEDataOnly">TRUE</ARG>
    <ARG NAME="SWESetNoTemp">TRUE</ARG>
  </CMD>
</EXEC>
```

### Navigating to a Screen

You use the GotoPageTab command to navigate to a specific screen. The Web application returns either an XML or HTML response containing data about the screen's views and applets. For a complete list of the screen names to which you can navigate, see [Table 15](#).

```
<?xml version="1.0" encoding="UTF-8"?>
<EXEC PATH="/callcenter/start.swe">
  <CMD NAME="SWECmd" VALUE="GotoPageTab">
    <ARG NAME="SWEScreen">screen name</ARG>
    <ARG NAME="SWESetMarkup">XML</ARG>
    <ARG NAME="SWEDataOnly">TRUE</ARG>
    <ARG NAME="SWESetNoTemp">TRUE</ARG>
  </CMD>
</EXEC>
```

## Navigating Within a Screen

When you use the `InvokeMethod` to execute an XML command, you must also indicate the view and the applet that you want to access. For example, you may want to modify or add a record. To add a record, you must first issue the `NewRecord` command, and then you must indicate to which view and applet you want the record to be added. To perform an action on a screen, you must navigate to the object within the screen that is to receive the action. The following two arguments are used to navigate within a screen:

- `SWEView`
- `SWEApplet`

For a complete list of the view and applet names to which you can navigate, see [Table 15](#). The example below details how to specify the view and applet:

```
<?xml version="1.0" encoding="UTF-8"?>
<EXEC PATH="/callcenter/start.swe">
<CMD NAME="SWECmd" VALUE="InvokeMethod">
<ARG NAME="SWEMethod">method name</ARG>
<ARG NAME="SWEView">view name</ARG>
<ARG NAME="SWEApplet">applet name</ARG>
<ARG NAME="SWESetMarkup">XML</ARG>
<ARG NAME="SWEDataOnly">TRUE</ARG>
<ARG NAME="SWESetNoTemp">TRUE</ARG>
</CMD>
</EXEC>
```

## Querying Items

To successfully perform a query, you must first navigate to a screen that allows queries. You must then send two separate requests to the eBusinessSWEXML application. The first request executes the Create New Query action, and the second executes the Execute Query action.

### NewQuery

```
<?xml version="1.0" encoding="UTF-8"?>
<EXEC PATH="/callcenter/start.swe">
  <CMD NAME="SWECmd" VALUE="InvokeMethod">
    <ARG NAME="SWEMethod">NewQuery</ARG>
    <ARG NAME="SWEView">view name</ARG>
    <ARG NAME="SWEApplet">applet name</ARG>
    <ARG NAME="SWESetMarkup">XML</ARG>
    <ARG NAME="SWEDataOnly">TRUE</ARG>
    <ARG NAME="SWESetNoTemp">TRUE</ARG>
  </CMD>
</EXEC>
```

### ExecuteQuery

In the ExecuteQuery command block, you must include an `<ARG>` tag. The tag must include a NAME parameter to identify the column (the field you want to search), and a value to indicate the search criteria.

```
<?xml version="1.0" encoding="UTF-8"?>
<EXEC PATH="/callcenter/start.swe"
  <CMD NAME="SWECmd" VALUE="InvokeMethod">
    <ARG NAME="SWEMethod">ExecuteQuery</ARG>
    <ARG NAME="SWEView">view name</ARG>
    <ARG NAME="SWEApplet">applet name</ARG>
    <ARG NAME="column name">search criteria</ARG>
    <ARG NAME="SWESetMarkup">XML</ARG>
```

```

<ARG NAME="SWEDataOnly">TRUE</ARG>
<ARG NAME="SWESetNoTemp">TRUE</ARG>
</CMD>
</EXEC>

```

The auction items that match the query are returned in the response. The returned payload contains complete lot names and IDs for each item.

---

**TIP:** Each row (or record) within a response contains an ID that uniquely identifies it. You can use a row ID as a parameter in a query to selectively single out a record so that you can modify or delete it.

---

## Adding Records

To successfully add a record to a list, you must first navigate to a screen that allows records to be inserted. Then, you must send two separate requests to the SWEXML application. The first request executes the New Record action. The second executes the WriteRecord action.

NewRecord

In a NewRecord command block, you use `<ARG >` tags to indicate the view and applet to which you want to add the NewRecord.

```

<?xml version="1.0" encoding="UTF-8"?>
<EXEC PATH="/callcenter/start.swe">
<CMD NAME="SWECmd" VALUE="InvokeMethod">
<ARG NAME="SWEMethod">NewRecord</ARG>
<ARG NAME="SWEView">view name</ARG>
<ARG NAME="SWEApplet">applet name</ARG><
ARG NAME="SWESetMarkup">XML</ARG>
<ARG NAME="SWEDataOnly">TRUE</ARG>
<ARG NAME="SWESetNoTemp">TRUE</ARG>
</CMD>
</EXEC>

```

### WriteRecord

In a WriteRecord command block, you must include an < ARG > tag for the row id of the record (SWERowID) and another < ARG > tag to indicate that the row id is required for the operation (SWEReqRowId).

```
<?xml version="1.0" encoding="UTF-8"?>
<EXEC PATH="/callcenter/start.swe">
  <CMD NAME="SWECmd" VALUE="InvokeMethod">
    <ARG NAME="SWEMethod">WriteRecord</ARG>
    <ARG NAME="SWEReqRowId">1</ARG>
    <ARG NAME="SWEView">view name</ARG>
    <ARG NAME="SWERowId">row id of record to be saved</ARG>
    <ARG NAME="SWEApplet">applet name</ARG>
    <ARG NAME="SWESetMarkup">XML</ARG>
    <ARG NAME="SWEDataOnly">TRUE</ARG>
    <ARG NAME="SWESetNoTemp">TRUE</ARG>
  </CMD>
</EXEC>
```

### Modifying Records

To successfully modify a record using XML, you must first navigate to a screen that allows records to be modified. Then, the following four requests must be sent separately to the SWEXML application:

- 1 Activate a new query.
- 2 Execute the query.
- 3 Activate the edit record method.

#### 4 Write the record.

---

**NOTE:** When modifying a record, you should use a primary key (such as a row ID) as the parameter for the query. This makes sure that only one record is returned and selected in the response. If you do not use a primary key to perform the query, several records may be returned in the response. There is a chance that the record you want to update is not the one selected.

---

### NewQuery

When you modify a record, you must first execute a query to find the record you want to modify. The records that are returned as a result of the query are then accessible through XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<EXEC PATH="/callcenter/start.swe">
<CMD NAME="SWECmd" VALUE="InvokeMethod">
<ARG NAME="SWEMethod">NewQuery</ARG>
<ARG NAME="SWEView">view name</ARG>
<ARG NAME="SWEApplet">applet name</ARG>
<ARG NAME="SWESetMarkup">XML</ARG>
<ARG NAME="SWEDataOnly">TRUE</ARG>
<ARG NAME="SWESetNoTemp">TRUE</ARG>
</CMD>
</EXEC>
```

### ExecuteQuery

When you use the ExecuteQuery command block in an effort to modify a record, you must include an `< ARG >` tag that identifies the primary key of the record you want to modify. This makes sure that the query returns only one record, which is automatically selected. You can then use the EditRecord command to update the selected record.

```
<?xml version="1.0" encoding="UTF-8"?>
<EXEC PATH="/callcenter/start.swe">
```

```
<CMD NAME="SWECmd" VALUE="InvokeMethod">
<ARG NAME="SWEMethod">ExecuteQuery</ARG>
<ARG NAME="SWEView">view name</ARG>
<ARG NAME="SWEApplet">applet name</ARG>
<ARG NAME="primary key column name">primary key value</ARG>
<ARG NAME="SWESetMarkup">XML</ARG>
<ARG NAME="SWEDataOnly">TRUE</ARG>
<ARG NAME="SWESetNoTemp">TRUE</ARG>

</CMD>

</EXEC>
```

### EditRecord

After executing the query the eBusiness screen is populated with the record you want to modify. You use the EditRecord to access the record.

---

**NOTE:** If you do not use a primary key to perform the query, several records may be returned in the response.

---

```
<?xml version="1.0" encoding="UTF-8"?>
<EXEC PATH="/callcenter/start.swe">
<CMD NAME="SWECmd" VALUE="InvokeMethod">
<ARG NAME="SWEMethod">EditRecord</ARG>
<ARG NAME="SWEView">view name</ARG>
<ARG NAME="SWEApplet">applet name</ARG>
<ARG NAME="column name1">field value</ARG>
<ARG NAME="column name2">field value</ARG>
...
<ARG NAME="column nameN">field value</ARG>
<ARG NAME="SWESetMarkup">XML</ARG>
<ARG NAME="SWEDataOnly">TRUE</ARG>
```

```
<ARG NAME="SWESetNoTemp">TRUE</ARG>
</CMD>
</EXEC>
```

## WriteRecord

In a WriteRecord command block, you must include an `< ARG >` tag for the row id of the record (SWERowID) and an argument to indicate the row id is required for the operation (SWEReqRowId).

```
<?xml version="1.0" encoding="UTF-8"?>
<EXEC PATH="/callcenter/start.swe">
<CMD NAME="SWECmd" VALUE="InvokeMethod">
<ARG NAME="SWEMethod">WriteRecord</ARG>
<ARG NAME="SWEReqRowId">1</ARG>
<ARG NAME="SWEView">view name</ARG>
<ARG NAME="SWERowId">row id of record to be saved</ARG>
<ARG NAME="SWEApplet">applet name</ARG>
<ARG NAME="SWESetMarkup">XML</ARG>
<ARG NAME="SWEDataOnly">TRUE</ARG>
<ARG NAME="SWESetNoTemp">TRUE</ARG>
</CMD>
</EXEC>
```

## Deleting Records

To successfully remove a record from the database, you must first navigate to a screen that allows records to be deleted. Then, the following three requests must be sent separately to the eBusinessSWEXML application:

- 1 Activate a new query.
- 2 Execute the query.

### 3 Delete the selected record.

---

**NOTE:** When deleting a record, you should use a primary key (such as a row ID) as the parameter for the query. This makes sure that only one record is returned and selected in the response. If you do not use a primary key to perform the query, several records may be returned in the response. There is a chance that the record you want to delete is not the one selected.

---

### NewQuery

When you delete a record, you must first execute a query to find the record you want to delete. You should use search criteria, such as a primary key, to make sure that the query returns only one record.

```
<?xml version="1.0" encoding="UTF-8"?>
<EXEC PATH="/callcenter/start.swe">
<CMD NAME="SWECmd" VALUE="InvokeMethod">
<ARG NAME="SWEMethod">NewQuery</ARG>
<ARG NAME="SWEView">view name</ARG>
<ARG NAME="SWEApplet">applet name</ARG>
<ARG NAME="SWESetMarkup">XML</ARG>
<ARG NAME="SWEDataOnly">TRUE</ARG>
<ARG NAME="SWESetNoTemp">TRUE</ARG>
</CMD>
</EXEC>
```

### ExecuteQuery

When you use the ExecuteQuery command block in an effort to delete a record, you must include an `< ARG >` tag that identifies the primary key of the record you want to delete. This makes sure that the query returns only one record, which is automatically selected. You can then use the DeleteRecord command to delete the selected record.

```
<?xml version="1.0" encoding="UTF-8"?>
<EXEC PATH="/callcenter/start.swe">
```

```

<CMD NAME="SWECmd" VALUE="InvokeMethod">
<ARG NAME="SWEMethod">ExecuteQuery</ARG>
<ARG NAME="SWEView">view name</ARG>
<ARG NAME="SWEApplet">applet name</ARG>
<ARG NAME="primary key column name">primary key value</ARG>
<ARG NAME="SWESetMarkup">XML</ARG>
<ARG NAME="SWEDataOnly">TRUE</ARG>
<ARG NAME="SWESetNoTemp">TRUE</ARG>
</CMD>
</EXEC>

```

## DeleteRecord

You use `<ARG>` tags to indicate the view and applet that contain the selected record you want to delete.

```

<?xml version="1.0" encoding="UTF-8"?>
<EXEC PATH="/callcenter/start.swe">
<CMD NAME="SWECmd" VALUE="InvokeMethod">
<ARG NAME="SWEMethod">DeleteRecord</ARG>
<ARG NAME="SWEView">view name</ARG>
<ARG NAME="SWEApplet">applet name</ARG>
<ARG NAME="SWESetMarkup">XML</ARG>
<ARG NAME="SWEDataOnly">TRUE</ARG>
<ARG NAME="SWESetNoTemp">TRUE</ARG>
</CMD>
</EXEC>

```

## Picking Records

To pick a value from a pick list and then save the value in the database, first you need to navigate to a screen and then submit three requests:

- 1 Navigate to a screen.
- 2 Get a pick list.
- 3 Get the RowId of the record to pick.
- 4 Write the record to the database.

### GotoPageTab

First you need to navigate to a screen. For example:

```
<EXEC PATH="/callcenter/start.swe">
<CMD VALUE="GotoPageTab" NAME="SWECmd">
<ARG NAME="SWEScreen">Accounts Screen</ARG>
<ARG NAME="SWENeedContext">>false</ARG>
<ARG NAME="SWEBID">-1</ARG>
</CMD>
<INFO NAME="SWEC">12</INFO>
</EXEC>
```

### EditField

To return the pick list using the EditField method, you must define arguments that identify the applet, view, and field on which the pick list is based. For example:

```
<EXEC PATH="/callcenter/start.swe">
<CMD VALUE="InvokeMethod" NAME="SWECmd">
<ARG NAME="SWEApplet">Account Entry Applet</ARG>
<ARG NAME="SWEW">0</ARG>
<ARG NAME="SWEView">Account List View</ARG>
<ARG NAME="SWERowId">1-6</ARG>
<ARG NAME="SWEField">Currency</ARG>
<ARG NAME="SWEDIC">>true</ARG>
<ARG NAME="SWENeedContext">>true</ARG>
<ARG NAME="SWEH">0</ARG>
```

```
<ARG NAME="SWEReqRowId">1</ARG>
<ARG NAME="SWESP">true</ARG>
<ARG NAME="SWEMethod">EditField</ARG>
</CMD>
<INFO NAME="SWEC">9</INFO>
</EXEC>
```

## PickRecord

The PickRecord method returns the RowId of the record to be picked. For example:

```
<EXEC PATH="/callcenter/start.swe">
<CMD VALUE="InvokeMethod" NAME="SWECmd">
<ARG NAME="SWEApplet">Currency Pick Applet</ARG>
<ARG NAME="SWEView">Account List View</ARG>
<ARG NAME="SWERowId">0-5129</ARG>
<ARG NAME="SWENeedContext">>false</ARG>
<ARG NAME="SWEReqRowId">1</ARG>
<ARG NAME="SWEP">14_Account Entry Applet9_EditField3_1-68_Currency1_1</ARG>
<ARG NAME="SWEMethod">PickRecord</ARG>
</CMD>
<INFO NAME="SWEC">1</INFO>
</EXEC>
```

---

**NOTE:** The value for the SWEP argument can be found in the XML response from EditField method.

---

## WriteRecord

The WriteRecord method writes the record to the database. For example:

```
<EXEC PATH="/callcenter/start.swe">
<CMD VALUE="InvokeMethod" NAME="SWECmd">
```

```
<ARG NAME="SWEApplet">Account Entry Applet</ARG>
<ARG NAME="SWEView">Account List View</ARG>
<ARG NAME="SWERowId">1-6</ARG>
<ARG NAME="SWENeedContext">true</ARG>
<ARG NAME="SWEReqRowId">1</ARG>
<ARG NAME="SWEMethod">WriteRecord</ARG>
</CMD>
<INFO NAME="SWEC">2</INFO>
</EXEC>
```

## SWE API

This section contains reference information about SWE commands, methods, and arguments.

## SWE Commands

Table 16 provides a list of commonly used SWE Commands.

**Table 16. SWE Commands**

Supported Values	Short Format	Description	Required Args (with Description)	Optional Args (with Description)
CanInvokeMethod For a list of commonly used methods, see <a href="#">Table 17</a> .	C	<p>Checks whether a method can be invoked on an applet, a business service, a buscomp, or the SWE application.</p> <p>Called only when OM is in High Interactivity mode.</p> <p>The optional <b>SWEService</b>, <b>SWEBusComp</b>, and <b>SWEApplet</b> arguments are used to specify the Siebel object that the method should be invoked on. If none of these are specified, SWE will check the CanInvokeMethod state of the method on the SWE application object, which currently supports a limited set of InvokeMethod, such as Logoff, SortOrder, SaveQuery, and SaveQueryAs.</p>	<b>SWEMethod</b> - name of the method.	<p><b>SWEService</b> - name of the business service to check whether the method can be invoked.</p> <p><b>SWEBusComp</b> - name of the business component to check whether the method can be invoked.</p> <p><b>SWEApplet</b> - name of the applet to check whether the method can be invoked.</p>

**Table 16. SWE Commands**

Supported Values	Short Format	Description	Required Args (with Description)	Optional Args (with Description)
ExecuteLogin	Xlg	Executes login for a user.	<b>SWEUserName</b> - user name. <b>SWEPassword</b> - password.	None
ExecuteNamedQuery	Xnq	Executes a PDQ.	<b>SWEView</b> - name of the view. <b>SWEQueryName</b> - name of the PDQ.	None
GotoPage	Gp	Goes to a Siebel Web page (this is the Web page object defined in Siebel Tools).	<b>SWEPage</b> - name of the Web page.	None
GotoPageTab	Gt	Goes to a Siebel screen. Will show the default view for the screen.	<b>SWEScreen</b> - name of the screen.	None

Table 16. SWE Commands

Supported Values	Short Format	Description	Required Args (with Description)	Optional Args (with Description)
GotoView	Gv	<p>Goes to a Siebel view.</p> <p>If the <b>SWEPostnApplet</b> and <b>SWEPostnRowId</b> arguments are specified, it will execute a search for the specified rowId in the specified applet.</p> <p>If <b>SWEQMApplet</b> and <b>SWEQMMethod</b> arguments are specified, it will invoke the method after going to the view.</p>	<b>SWEView</b> - name of the view.	<p><b>SWEKeepContext</b> - if TRUE, keeps the current business object context, when requesting to a view based on the same business object.</p> <p><b>SWEPostnApplet</b> - name of the applet on which the search should be executed.</p> <p><b>SWEPostnRowId</b> - row Id to search for.</p> <p><b>SWEQMApplet</b> - name of the QueueMethod applet. This is the applet where the method (as specified in <b>SWEQMMethod</b>) should be invoked after going to the view.</p> <p><b>SWEQMMethod</b> - name of the QueueMethod method to be invoked. You can invoke only one method.</p> <p><b>SWEQMArgs</b> - arguments of the QueueMethod method.</p>

**Table 16. SWE Commands**

Supported Values	Short Format	Description	Required Args (with Description)	Optional Args (with Description)
InvokeMethod For a list of commonly used methods, see <a href="#">Table 17</a> .	Inv	<p>Invokes a method on an applet, a business service, a business component, or the SWE application.</p> <p>The optional <b>SWEService</b>, <b>SWEBusComp</b>, and <b>SWEApplet</b> arguments are used to specify the Siebel object on which the method should be invoked. If none of these are specified, SWE will invoke on the SWE application object, which currently supports a limited set of InvokeMethod such as Logoff, SortOrder, SaveQuery, and SaveQueryAs.</p>	<b>SWEMethod</b> - name of the method.	<p><b>SWEService</b> - name of the business service to invoke the method.</p> <p><b>SWEBusComp</b> - name of the business component to invoke the method.</p> <p><b>SWEApplet</b> - name of the applet to invoke the method.</p> <p><b>SWEView</b> - name of the view to invoke the method</p>
LoadService		Loads a business service on the server side.	<b>SWEService</b> - name of the business service to load.	None
Login	Lg	Loads the login view or login page. SWE first looks at the Acknowledgment Web View property of the application object in the repository for the login view to show. If not specified, the default is the "Acknowledgment Web Page" property to show the login page.	None	None

**Table 16. SWE Commands**

Supported Values	Short Format	Description	Required Args (with Description)	Optional Args (with Description)
Logoff	Bye	Executes the database logoff, then shows the logoff view or page. SWE first looks at the Logoff Acknowledgment Web Page property of the application object in the repository for the login page to show. If none is specified, SWE will show the login view or login page, depending on how you log in.	None	None
ReloadCT		Reloads personalization info. SWE loads the initial personalization on startup, and when the personalization rules are changed, SWE does not update the info automatically since there is cost in performance, so SWE provides this command to reload the info.	None	None

**NOTE:** The SWEAC command allows users to string two SWE commands in a single request. For example the following URL does a SWECmd = ExecuteLogin, and then a SWEAC = GotoPageTab.

```
SWECmd=ExecuteLogin&SWEUserName=joe&SWEPasswd=passwd&SWEAC=SWECmd=GotoPageTab&SWEScreen=Accounts+Screen&SWEReloadFrames=1.
```

### SWE Methods

The InvokeMethod command allows you to invoke methods on a applet, business component, business service, or application. [Table 17](#) lists SWE methods commonly used with the InvokeMethod SWE command.

**Table 17. SWE Methods**

Supported Values	Description	Required Args (with Description)	Optional Args (with Description)
CollapseTreeItem	Used in a tree control to collapse an expanded item on the tree.	<b>SWETreeItem:</b> Specify the path of the item relative to root. The path is a string of the form n.n.n.n...where <i>n</i> is an index of an item within its level. The index starts from 1. Example: 1.1.2. <b>SWEView:</b> Name of the view. <b>SWEApplet:</b> Name of the applet.	None
CopyRecord	Performs initialization, then calls CopyRecord on the business component.	None	None
CreateRecord	Performs initialization, then calls NewRecord on the business component.	None	None
DeleteQuery	Deletes a named query.	<b>SweNamedQueries:</b> Specify the name of the named query to be deleted.	None
DeleteRecord	Deletes a record.	None	None
Drilldown	Drills down on the field as specified in the argument <b>SWEField</b> .	<b>SWEField:</b> Specify the name of the applet field that you want to drilldown on. The drilldown information is specified in the repository.	None

Table 17. SWE Methods

Supported Values	Description	Required Args (with Description)	Optional Args (with Description)
EditRecord	Edits a record.	<b>SWESeq:</b> Specify the sequence number of the Edit template. You can have many Edit templates for an applet in Siebel Tools, each identified by the sequence number.	List of arguments with name and value, where the name specifies the field name and the value specifies the field query specification. Will set field query specification before executing the query.
ExecuteQuery	Executes a query. The query specification of the fields is specified in the list of arguments.	None	
ExpandTreeItem	Used in a tree control to expand an item on the tree.	<b>SWETreeItem:</b> Specify the path of the item relative to root. The path is a string of the form n.n.n.n...where <i>n</i> is an index of an item within its level. The index starts from 1. Example: 1.1.2. <b>SWEView:</b> Name of the view. <b>SWEApplet:</b> Name of the applet.	None
GotoFirstSet	Goes to the first set of records. The number of rows in a set is specified in the repository.	None	None
GotoLastSet	Goes to the last set of records.	None	None
GotoNextSet	Goes to the next set of records.	None	None
GotoPreviousSet	Goes to the previous set of records.	None	None

**Table 17. SWE Methods**

Supported Values	Description	Required Args (with Description)	Optional Args (with Description)
GotoView	<p>Goes to a Siebel view.</p> <p>If the <b>SWEPostnApplet</b> and <b>SWEPostnRowId</b> arguments are specified, will execute a search for the specified rowId in the specified applet.</p> <p>If <b>SWEQMApplet</b> and <b>SWEQMMethod</b> arguments are specified, will invoke the method after going to the view.</p>	<b>SWETargetView</b> - name of the view.	<p><b>SWEKeepContext</b> - if TRUE, keeps the current business object if going to a view that uses the same business object.</p> <p><b>SWEPostnApplet</b> - name of the applet that the search should be executed on.</p> <p><b>SWEPostnRowId</b> - rowId to search for.</p> <p><b>SWEQMApplet</b> - name of the QueueMethod applet. This is the applet where the method (as specified in <b>SWEQMMethod</b>) should be invoked after going to the view.</p> <p><b>SWEQMMethod</b> - name of the QueueMethod method. The method to be invoked. You can invoke only one method.</p> <p><b>SWEQMArgs</b> - arguments of the QueueMethod method.</p>
Indent	For a hierarchical applet, moves the current record down the hierarchy by one level.	None	None
MoveDown	For a hierarchical applet, moves the current record down the hierarchy within the same level.	None	None
MoveUp	For a hierarchical applet, moves the current record up the hierarchy within the same level.	None	None

Table 17. SWE Methods

Supported Values	Description	Required Args (with Description)	Optional Args (with Description)
NewQuery	Begins a new query.	None	None
NewRecord	If the applet has an association applet, shows the association popup applet. Otherwise, creates a new record.	None	None
NextTreeItem	Used in a tree control to scroll the tree to the next set of record.	<b>SWETreeItem:</b> Specifies the path of the item relative to root. The path is a string of the form n.n.n.n... where <i>n</i> is an index of an item within its level. The index starts from 1. Example: 1.1.2. <b>SWEView:</b> Name of the view. <b>SWEApplet:</b> Name of the applet.	None
Outdent	For a hierarchical applet, moves the current record down the hierarchy by one level.	None	None
PickNone	Makes sure the parent applet field has nothing picked from the pick applet.	None	None
PickRecord	Picks the current row in a pick applet.	None	None
PositionOnRow	Positions the record as specified in the arguments <b>SWERowIds</b> and <b>SWERowId</b> . If no arguments, does nothing.	None	<b>SWERowIds:</b> a string specifying the row Id of the parent buscomps. <b>SWERowId:</b> the row Id of the record to position to.

**Table 17. SWE Methods**

Supported Values	Description	Required Args (with Description)	Optional Args (with Description)
PostChanges	Sets the field values as specified in the list of arguments to the record being created or edited.	None	List of arguments with name and value where the name specifies the field name and the value specifies the field value. Will set these field values before committing the record.
PreviousTreeItem	Used in a tree control to scroll the tree to the previous set of records.	<b>SWETreeItem:</b> Specify the path of the item relative to root. The path is a string of the form n.n.n.n...where <i>n</i> is an index of an item within its level. The index starts from 1. Example: 1.1.2. <b>SWEView:</b> Name of the view. <b>SWEApplet:</b> Name of the applet.	None
RefineQuery	Keeps the current field query specification and queries again.	None	None
SaveQueryAs	Saves the current query as a named query. The name is specified in the argument <b>_SweNamedQueries</b> .	<b>SweNamedQueries:</b> Specify the name to save the query as.	None

Table 17. SWE Methods

Supported Values	Description	Required Args (with Description)	Optional Args (with Description)
SelectTreeItem	Used in a tree control to select an item of the tree.	<b>SWETreeItem</b> : Specifies the path of the item relative to root. The path is a string of the form n.n.n.n... where <i>n</i> is an index of an item within its level. The index starts from 1. Example: 1.1.2. <b>SWEView</b> : Name of the view. <b>SWEApplet</b> : Name of the applet.	None
SortAscending	Sorts the field as specified in the argument <b>SWEField</b> in ascending order.	<b>SWEField</b> : Specifies the name of the applet field that you want to sort in ascending order.	None
SortDescending	Sorts the field as specified in the argument <b>SWEField</b> in descending order.	<b>SWEField</b> : Specifies the name of the applet field that you want to sort in descending order.	None
ToggleTo	Toggles to a different toggle applet.	<b>SWESeq</b> : Sequence number of the togglet applet to toggle to.	None
UndoRecord	Undoes a record that is being created or edited.	None	None
WriteRecord	Commits a record that is being created or edited.	<b>SWERowId</b> : Is the Row ID of the record to be saved. <b>SWEReqRowId</b> : Indicates that the Row ID is required in the operation.	None

## SWE Arguments

Table 18 lists some commonly used SWE arguments.

**Table 18. SWE Arguments**

URL Argument	Short Format	Description	Usage	Examples
SWECCount	C	Dynamically generates an index number for each hyperlink for the purpose of bookmarking each request.	SWEC = n, where n is a positive integer number. < ARG NAME = "SWEC" > n < / ARG >	SWEC = 1, or < ARG NAME = "SWEC" > 1 < / ARG >
SWEDataOnly		Discards all UI content (including anchors) if set to TRUE.	SWEDataOnly = {TRUE   FALSE} < ARG NAME = "SWEDataOnly" > TRUE FALSE < / ARG >	SWEDataOnly = TRUE < ARG NAME = "SWEDataOnly" > TRUE < / ARG >
SWEEExclude		Uses the comma-separated UI element names specified as the value of the parameter to exclude UI elements from appearing in the output document.	SWEEExclude = "list of names". Names can be MENU, SCREENBAR, TOOLBAR, THREADBAR, PAGEITEM, VIEWBAR. < ARG NAME = "SWEEcludet" > list of names < / ARG >	SWEEExclude = "MENU, SCREENBAR" < ARG NAME = "SWEEExclude" > MENU,SCREENBAR < / ARG >
SWEField	F	Specifies the name of the applet field.	SWEField = < field name > < ARG NAME = "SWEField" > field name < / ARG >	SWEField = Revenue < ARG NAME = "SWEField" > Revenue < / ARG >

Table 18. SWE Arguments

URL Argument	Short Format	Description	Usage	Examples
SWEFullRefresh		Forces a full refresh of the Siebel Web Client. Used by the High Interactivity client to send a SWE command to load the High Interactivity client completely. Typically used for session interleaving from a non-Siebel session to the Siebel High Interactivity client application.	SWEFullRefresh = {TRUE   FALSE} < ARG NAME = "SWEFullRefresh" > TRUE FALSE < /ARG >	SWEFullRefresh = TRUE < ARG NAME = "SWEFullRefresh" > TRUE < /ARG >
SWEGetApplet		This parameter is used to filter the outbound XML document so only the applet named as the value of the parameter will be allowed in the output. All other document content will be discarded.	SWEGetApplet = < name of the applet > < ARG NAME = "SWEGetApplet" > name of the applet < / ARG >	SWEGetApplet = Account + List + Applet < ARG NAME = "SWEGetApplet" > Account List Applet < /ARG >
SWEGetPDQ		Discards all XML content and returns only PDQ list when set to TRUE.	SWEGetPDQ = {TRUE   FALSE} < ARG NAME = "SWEGetPDQ" > TRUE FALSE < /ARG >	
SWEKeepContext	Kx	Keeps the current business object if going to a view that uses the same business object, if set to TRUE.	SWEKeepContext = {TRUE   FALSE} < ARG NAME = "SWEKeepContext" > TRUE FALSE < / ARG >	SWEKeepContext = TRUE < ARG NAME = "SWEKeepContext" > TRUE < / ARG >

**Table 18. SWE Arguments**

URL Argument	Short Format	Description	Usage	Examples
SWENeedContext	Nct	Skips restoring the state of the view, applet, busobj, and buscomp when going back to a previously viewed page, if set to FALSE.  Default is TRUE for a view or applet and FALSE for a Web page.	SWENeedContext = {TRUE   FALSE} < ARG NAME = "SWENeedContext" > TRUE FALSE < / ARG >	SWENeedContext = TRUE < ARG NAME = "SWENeedContext" > TRUE < / ARG >
SWENoAnchor		Discards all anchors if set to TRUE.	SWENoAnchor = { TRUE   FALSE } < ARG NAME = "SWENoAnchor" > TRUE FALSE < / ARG >	SWENoAnchor = TRUE < ARG NAME = "SWENoAnchor" > TRUE < / ARG >
SWEReloadFrames	RF	Forces the reloading of all HTML frames when set to TRUE.	SWERF = {TRUE FALSE}, or < ARG NAME = "SWERF" > TRUE   FALSE < / ARG >	SWERF = TRUE, or < ARG NAME = "SWERF" > TRUE < / ARG >
SWEReqRowId	Rqr	Needs to position to the row specified in the argument SWERowI, if set to TRUE.	SWEReqRowId = { TRUE   FALSE } < ARG NAME = "SWEReqRowId" > TRUE FALSE < / ARG >	SWEReqRowId = TRUE < ARG NAME = "SWEReqRowId" > TRUE < / ARG >
SWERows	Rs	Specifies the number of rows to be used as an attribute of an HTML frameset.	SWERs = n, where n is a positive integer number. Or < ARG NAME = "SWERs" > n < / ARG >	SWERs = 1, or < ARG NAME = "SWERs" > 1 < / ARG >
SWERowId	R	The rowId of the record to position to.	SWERowId = < rowid > < ARG NAME = "SWERowId" > rowid < / ARG >	SWERowId = 12-XI46FG < ARG NAME = "SWERowId" > 12-XI46FG < / ARG >

**Table 18. SWE Arguments**

URL Argument	Short Format	Description	Usage	Examples
SWERowIds	Rs	A string specifying the rowId of the parent buscomps.	SWERowIds = < string of rowids > < ARG NAME = "SWERowId" > string of rowids < /ARG >	SWERowIds = SWERowId0%3d12-61W25L < ARG NAME = "SWERowId" > SWERowId = 12-61W25L < /ARG >
SWESetMarkup		Temporarily sets the markup language to use in the output document.	SWESetMarkup = < name of the markup language > < ARG NAME = "SWESetMarkup" > markup language < / ARG >	SWESetMarkup = HTML < ARG NAME = "SWESetMarkup" > HTML < /ARG >
SWESetNoTempl		Disables the use of templates during the generation of the outbound document.	SWESetNoTempl = {TRUE   FALSE} < ARG NAME = "SWESetNoTempl" > TRUE FALSE < /ARG >	SWESetNoTempl = TRUE < ARG NAME = "SWESetNoTempl" > TRUE < /ARG >
SWESetRowCnt		Temporarily sets the workset size or row number of list applets in the view.	SWESetRowCnt = < number of list rows > < ARG NAME = "SWESetRowCnt" > number of list rows < / ARG >	SWESetRowCnt = 50 < ARG NAME = "SWESetRowCnt" > number of list rows < /ARG >
SWEXslStyleSheet		Specifies the name of the XSLT stylesheet to use to perform the XSLT on the XML output document.	SWEXslStyleSheet = < stylesheet name > . The stylesheet needs to be in the application's webtempl directory. < ARG NAME = "SWEXslStyleSheet" > name of the XSLT stylesheet < /ARG >	SWEXslStyleSheet = ui.xsl < ARG NAME = "SWEXslStyleSheet" > ui.xsl < / ARG >

## Document Type Definition

This section lists Document Type Definitions (DTD) for the inbound and outbound documents used with the XML Web Interface.

### Inbound DTD

```
<!-- Copyright (c) 2001 Siebel Systems, Inc. -->

<!ELEMENT EXEC (CMD, INFO*) >
<!ATTLIST EXEC

ATTRCDATA#IMPLIED
PATHCDATA#IMPLIED
TARGETCDATA#IMPLIED
>
<!ELEMENT CMD (ARG*) >
<!ATTLIST CMD

NAMECDATA#REQUIRED
VALUECDATA#REQUIRED
>
<!ELEMENT ARG (#PCDATA) >
<!ATTLIST ARG
NAMECDATA#REQUIRED
>
<!ELEMENT INFO (#PCDATA) >
<!ATTLIST INFO

NAMECDATA#REQUIRED
>
```

### Outbound DTD

```
<!-- Copyright (c) 2001 Siebel Systems, Inc. -->

<!ELEMENT APPLICATION (ERROR*, (USER_AGENT?, NAVIGATION_ELEMENTS*, (SCREEN
| APPLLET | FORM | PDQ_BAR)* ), ERROR*) >
<!ATTLIST APPLICATION

NAME CDATA #REQUIRED
>

<!ELEMENT USER_AGENT EMPTY>
<!ATTLIST USER_AGENT

MARKUPCDATA#REQUIRED
TYPECDATA#IMPLIED
>

<!ELEMENT NAVIGATION_ELEMENTS (MENU*,
TOOL_BAR*,
SCREEN_BAR*,
```

```

THREAD_BAR*,
VIEW_BAR*,
PAGE_ITEM*) >

<!ELEMENT MENU (MENU_ITEM | ERROR)* >
<!ATTLIST MENU
    NAME CDATA #REQUIRED
>

<!ELEMENT MENU_ITEM (#PCDATA | ANCHOR | MENU_ITEM | ERROR)* >
<!ATTLIST MENU_ITEM
    NAME CDATA #IMPLIED
    ENABLED (TRUE | FALSE) #IMPLIED
    TYPE CDATA #IMPLIED
>

<!ELEMENT ANCHOR ((CMD, INFO*) | ERROR*) >
<!ATTLIST ANCHOR
    ATTR CDATA #IMPLIED
    PATH CDATA#IMPLIED
    TARGET CDATA#IMPLIED
>

<!ELEMENT CMD (ARG*) >
<!ATTLIST CMD
    NAMECDATA #REQUIRED
    VALUECDATA#REQUIRED
>

<!ELEMENT ARG (#PCDATA) >
<!ATTLIST ARG
    NAME CDATA #REQUIRED
>

<!ELEMENT INFO (#PCDATA) >
<!ATTLIST INFO
    NAME CDATA #REQUIRED
>

<!ELEMENT TOOL_BAR (TOOL_ITEM | ERROR)* >
<!ATTLIST TOOL_BAR

```

```
NAMECDATA #REQUIRED
PATHCDATA#IMPLIED
>

<!ELEMENT  TOOL_ITEM                (#PCDATA | ANCHOR | ERROR)* >
<!ATTLIST  TOOL_ITEM

NAMECDATA#REQUIRED
TYPECDATA#REQUIRED

ATTRCDATA#IMPLIED
MAX_LENGTH CDATA#IMPLIED
>

<!ELEMENT  SCREEN_BAR              (SCREEN_TAB | VIEW_BAR | ERROR)* >
<!ELEMENT  SCREEN_TAB              (#PCDATA | VIEW_BAR | ANCHOR | ERROR)* >
<!ATTLIST  SCREEN_TAB

NAMECDATA#REQUIRED
ACTIVE(TRUE | FALSE) "FALSE"

    CAPTION      CDATA              #IMPLIED
>

<!ELEMENT  THREAD_BAR              (THREAD | ERROR)* >
<!ELEMENT  THREAD                  (#PCDATA | ANCHOR | ERROR)* >
<!ATTLIST  THREAD

    TITLE        CDATA              #REQUIRED
>

<!ELEMENT  VIEW_BAR                (VIEW_TAB | ERROR)* >
<!ATTLIST  VIEW_BAR

MODECDATA#IMPLIED
SCREENCDATA #IMPLIED
TYPECDATA#IMPLIED
>

<!ELEMENT  VIEW_TAB                (#PCDATA | ANCHOR | ERROR)* >
<!ATTLIST  VIEW_TAB

NAMECDATA#REQUIRED
SELECTED(TRUE | FALSE) "FALSE"
TITLECDATA #IMPLIED
>

<!ELEMENT  PAGE_ITEM               (#PCDATA | ANCHOR | ERROR)* >
<!ATTLIST  PAGE_ITEM

NAMECDATA#REQUIRED
ATTRCDATA#IMPLIED
CAPTIONCDATA#IMPLIED
```

```

TYPECDATA#REQUIRED
>

<!ELEMENT SCREEN (VIEW | ERROR*) >
<!ATTLIST SCREEN

NAMECDATA#REQUIRED
ACTIVE(TRUE | FALSE)"FALSE"
CAPTIONCDATA#IMPLIED
>

<!ELEMENT VIEW (SUB_VIEW_BAR | PDQ_BAR | APPLETT | IMG | FORM | ERROR)* >
<!ATTLIST VIEW

NAME CDATA #REQUIRED
ACTIVE (TRUE | FALSE) "FALSE"
CATEGORY CDATA #IMPLIED
TITLE CDATA #IMPLIED
>

<!ELEMENT APPLETT (FORM | CONTROL | CALENDAR | TREE | (LIST |
(RS_HEADER, RS_DATA)) | SORT_FIELD | APPLETTOGGLE | ERROR)* >
<!ATTLIST APPLETT

NAMECDATA#REQUIRED
ACTIVECDATA#IMPLIED
CLASSCDATA#IMPLIED
IDCDATA#IMPLIED
MODECDATA#IMPLIED
NO_DELETE(TRUE | FALSE)"FALSE"
NO_EXEC_QUERY(TRUE | FALSE)"FALSE"
NO_INSERT(TRUE | FALSE)"FALSE"
NO_MERGE (TRUE | FALSE)"FALSE"
NO_UPDATE(TRUE | FALSE)"FALSE"
ROW_COUNTERCDATA#IMPLIED
TITLECDATA#IMPLIED
>

<!ELEMENT FORM ((CONTROL | CALENDAR | TREE | (LIST | (RS_HEADER,
RS_DATA)) | SORT_FIELD | APPLETTOGGLE | PDQ_BAR | SUB_VIEW_BAR)* | ERROR)* >
<!ATTLIST FORM

NAME CDATA #IMPLIED
ACTION CDATA #IMPLIED
ATTR CDATA #IMPLIED
METHOD CDATA #IMPLIED
TARGET CDATA #IMPLIED
>

<!ELEMENT CONTROL (#PCDATA | IMG | ANCHOR | PICK_LIST | ERROR)* >
<!ATTLIST CONTROL

```

```
NAME          CDATA          #REQUIRED
ATTR          CDATA          #IMPLIED
CALCULATED    ( TRUE | FALSE ) "FALSE"
CAPTION       CDATA          #IMPLIED
DATATYPE      CDATA          #IMPLIED
ENABLED       ( TRUE | FALSE ) "FALSE"
FIELD         CDATA          #IMPLIED
FORMAT        CDATA          #IMPLIED
HIDDEN        ( TRUE | FALSE ) "FALSE"
HTML_TYPE     CDATA          #IMPLIED
ID            CDATA          #IMPLIED
MAX_LENGTH    CDATA          #IMPLIED
NUMBER_BASED  ( TRUE | FALSE ) "FALSE"
READ_ONLY     ( TRUE | FALSE ) "FALSE"
REQUIRED      ( TRUE | FALSE ) "FALSE"
REQUIRED_INDICATOR CDATA          #IMPLIED
SCALE         CDATA          #IMPLIED
TEXT_ALIGN    CDATA          #IMPLIED
TEXT_BASED    ( TRUE | FALSE ) "FALSE"
TYPE          CDATA          #IMPLIED
VARIABLE      CDATA          #IMPLIED
>

<!ELEMENT PICK_LIST          (OPTION | ERROR)* >
<!ATTLIST PICK_LIST

NAME          CDATA          #IMPLIED
ATTR          CDATA          #IMPLIED
VALUE         CDATA          #IMPLIED
>

<!ELEMENT OPTION            (#PCDATA | ERROR)* >
<!ATTLIST OPTION

CAPTION       CDATA          #IMPLIED
SELECTED      ( TRUE | FALSE ) "FALSE"
>

<!ELEMENT LIST              ((RS_HEADER, RS_DATA) | ALERT | ERROR*) >

<!ELEMENT RS_HEADER         (METHOD | COLUMN | ERROR)* >
<!ELEMENT RS_DATA           (ROW | ERROR)* >

<!ELEMENT METHOD            (#PCDATA | ANCHOR)* >
<!ATTLIST METHOD

NAME          CDATA          #REQUIRED
CAPTION       CDATA          #IMPLIED
FIELD         CDATA          #IMPLIED
```

```

>
<!ELEMENT COLUMN (METHOD | ERROR)* >
<!ATTLIST COLUMN
NAMECDATA#REQUIRED
CALCULATED(TRUE | FALSE)"FALSE"
DISPLAY_NAMECDATA#IMPLIED
DATATYPECDATA#IMPLIED
FIELDCDATA#IMPLIED
FORMATCDATA#IMPLIED
HIDDEN(TRUE | FALSE)"FALSE"
HTML_TYPECDATA#IMPLIED
IDCDATA#IMPLIED
LIST_EDITABLECDATA#IMPLIED
NUMBER_BASED(TRUE | FALSE)"FALSE"
READ_ONLY(TRUE | FALSE)"FALSE"
REQUIRED(TRUE | FALSE)"FALSE"
SCALECDATA#IMPLIED
TEXT_ALIGNCDATA#IMPLIED
TEXT_BASED(TRUE | FALSE)"FALSE"
TEXT_LENGTHCDATA#IMPLIED
TOTAL_REQUIRED(TRUE | FALSE)"FALSE"
TYPECDATA#IMPLIED
>
<!ELEMENT ROW (#PCDATA | FIELD | ERROR)* >
<!ATTLIST ROW
ROWIDCDATA#REQUIRED
SELECTED(TRUE | FALSE)"FALSE"
>
<!ELEMENT FIELD (#PCDATA | PICK_LIST | ANCHOR | ERROR)* >
<!ATTLIST FIELD
NAME CDATA #REQUIRED
VARIABLE CDATA #IMPLIED
>
<!ELEMENT TREE (ITEM | ERROR)* >
<!ATTLIST TREE
NAME CDATA #REQUIRED
>
<!ELEMENT ITEM (#PCDATA | ACTION | ITEM | ERROR)* >
<!ATTLIST ITEM
ATTR CDATA #IMPLIED
CAPTION CDATA #IMPLIED
PATH CDATA #REQUIRED
SELECTED (TRUE | FALSE) "FALSE"

```

```

TYPE          CDATA          #IMPLIED
>

<!ELEMENT ACTION          (#PCDATA | ANCHOR)* >
<!ATTLIST ACTION

ATTR          CDATA          #IMPLIED
TYPE          CDATA          #REQUIRED
>

<!ELEMENT CALENDAR        EMPTY>
<!ATTLIST CALENDAR

TITLE         CDATA          #IMPLIED
>

<!ELEMENT SORT_FIELD      (PICK_LIST | ERROR)* >
<!ATTLIST SORT_FIELD

NAME          CDATA          #REQUIRED
SEQUENCE     CDATA          #IMPLIED
>

<!ELEMENT APPLETTOGGLE    (TOGGLE_ITEM | ERROR)* >
<!ATTLIST APPLETTOGGLE

    TYPE          CDATA          #IMPLIED
>

<!ELEMENT TOGGLE_ITEM     (#PCDATA | ANCHOR | ERROR)* >
<!ATTLIST TOGGLE_ITEM

APPLET_NAME  CDATA          #REQUIRED
TITLE        CDATA          #IMPLIED
SELECTED     (TRUE | FALSE) "FALSE"
>

<!ELEMENT SUB_VIEW_BAR    (VIEW_TAB | ERROR)* >

<!ELEMENT PDQ_BAR         (PDQ | ERROR)* >

<!ELEMENT PDQ             (#PCDATA | ANCHOR | ERROR)* >
<!ATTLIST PDQ

NAME         CDATA          #REQUIRED
SELECTED     (TRUE | FALSE) "FALSE"
>

<!ELEMENT IMG             (#PCDATA) >
<!ATTLIST IMG

ALT          CDATA          #IMPLIED
SRC          CDATA          #IMPLIED
>
```

```
<!ELEMENT ERROR(#PCDATA | ERROR)* >  
<!ELEMENT ALERT (#PCDATA) >
```

## Manipulating Siebel XML with XSL Stylesheets and XSLT

SWE can perform embedded XSL transformation on outbound XML documents. In this way, you can generate outbound documents in the desired markup language or format directly from SWE, without requiring a middle-tier server to perform the transformation. To do so, application developers must provide the XSL stylesheets used for the transformation and specify the names of the stylesheets to SWE.

### Defining SWTs Stylesheet Tags

There are two ways in which you can request SWE to transform the outbound XML document into the desired format using XSLT. You can either pass in a query parameter `SWEXslStyleSheet = name-of-the-stylesheet`, or you can specify the stylesheets to use in the Siebel templates by means of the `< swe:xsl-stylesheet >` tag (see “[XML-Specific Template Tag](#)”).

### XML-Specific Template Tag

The XML-specific template tag looks like this:

```
< swe:xsl-stylesheet >
```

#### **Purpose:**

Specifies the name of the XSLT stylesheet to perform the XSLT on the XML output document. The stylesheet must reside in the application’s `webtempl` directory. There is only one `< swe:xsl-stylesheet >` tag for each view. If more than one `< swe:xslstylesheet >` tag is specified in the view, the last tag found gets used.

#### **Attributes:**

**name.** Specifies the name of the stylesheet.

**mode.** You can set the mode to either *process* or *embed*. When set to *process*, SWE performs XSLT processing on the XML output and sends the transformed document as the response back to the client. When set to *embed*, SWE inserts an XML processing instruction in the beginning of the XML document for external XSLT processing.

### Example

```
<swe:xsl-stylesheet name= "table.xsl" mode= "process" />
```

## Sample Stylesheet Code

The following XSL style sheet code sample is used to transform the WML-based Siebel Wireless application into HTML through the XML Web Interface. This code shows how a list view in the Wireless application is converted to HTML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="html" media-type="text/html"/>

<!-- This style sheet process the XML output for both the Splash screens and
standard views-->

<!-- ===== Root Document Processing =====>

<!-- Document Root-->

<xsl:template match="/">

<xsl:apply-templates select="//APPLICATION/SCREEN/VIEW/APPLET"></xsl:apply-
templates>

</xsl:template>

<!-- ===== View Processing =====>

<!-- List Base mode Template-->

<xsl:template match="APPLET">

<HTML>

<BODY>

<b>

<!-- Applet Title Label-->

<xsl:value-of select="CONTROL[@ID='1']"/>
```

```
<!-- for calendar title -->
<xsl:value-of select="CALENDAR/@TITLE"/>
</b>
<br></br>
<!-- XML No Record found and other alerts -->
<xsl:if test="string-length(ALERT)>0 and @CLASS='CSSFrameCalRerouteBase'">
<xsl:value-of select="ALERT"/>
<br></br>
</xsl:if>
<!-- Search and Title with data or other links -->
<xsl:apply-templates select="CONTROL[@ID=2 or @ID=3 or @ID=4 or @ID=5 or @ID=6 or
@ID=7 or @ID=8 or @ID=9]"/>
<!-- Separator line -->
<xsl:apply-templates select="CONTROL[@ID=1000]"/>
<!-- Display fields for list of records here-->
<xsl:apply-templates select="LIST"></xsl:apply-templates>
<xsl:if test="string-length(@ROW_COUNTER)>0">
<xsl:value-of select="@ROW_COUNTER"></xsl:value-of>
<br></br>
</xsl:if>
<!-- control link for New, Main Menu, etc.. -->
<xsl:apply-templates select="CONTROL[@ID]>=40 and @HTML_TYPE='Link'"/>
</BODY>
</HTML>
</xsl:template>
<!-- ===== Control and Link Processing =====>
<xsl:template match="CONTROL">
<xsl:choose>
    <xsl:when test="@HTML_TYPE='Link'">
```

```
<xsl:call-template name="build_simple_link"></xsl:call-template>

    </xsl:when>

<xsl:otherwise>

<xsl:value-of select="."></xsl:value-of><br></br>

</xsl:otherwise>

    </xsl:choose>

</xsl:template>

<xsl:template name="build_simple_link">

<xsl:variable name="link">

<xsl:apply-templates select="ANCHOR"></xsl:apply-templates>

</xsl:variable>

<xsl:element name="A">

<xsl:attribute name="HREF"><xsl:value-of select="$link"/></xsl:attribute>

<xsl:value-of select="@CAPTION"/>

</xsl:element>

<br/>

</xsl:template>

<!-- ===== List processing =====>

<!-- LIST Template builds a list of records -->

<xsl:template match="LIST">

<!-- first get the URL from the RS_HEADER element-->

<xsl:variable name="link">

<xsl:apply-templates select="RS_HEADER/METHOD[@NAME='Drilldown']"/>

</xsl:variable>

<!-- capture the URL before the SWERowId parameter-->

<xsl:variable name="link-prefix">

<xsl:value-of select="substring-before($link,'R=')"/>

</xsl:variable>
```

```
<!-- capture the URL after the SWERowId parameter-->
<xsl:variable name="link-suffix">
<xsl:value-of select="substring-after($link,'R=')"/>
</xsl:variable>
<!-- capture the field with the drilldown enabled - use later to build drilldown -->
<xsl:variable name="drilldowncontrol">
<xsl:value-of select="RS_HEADER/METHOD[@NAME='Drilldown']/@FIELD"></xsl:value-of>
</xsl:variable>
<!-- loop through the rows in the RS_DATA element -->
<xsl:for-each select="RS_DATA/ROW">
<!-- pickup the Row Id for the Row so we can rebuild the SWERowId URL parameter-->
<xsl:variable name="rowid">
<xsl:value-of select="@ROWID"/>
</xsl:variable>
<!-- loop through each field and control in the Row -->
<xsl:for-each select="FIELD|CONTROL">
<xsl:choose>
<!-- if the field is the drilldown field then create a link on the display data-->
<xsl:when test="@NAME = $drilldowncontrol">
<xsl:element name="A">
<xsl:attribute name="HREF">
<xsl:value-of select="concat(normalize-space($link-prefix),'R',$rowid,$link-
suffix)"/>&amp;F=<xsl:value-of select="@VARIABLE"/>
</xsl:attribute>
<xsl:value-of select="."/;></xsl:value-of>
</xsl:element>
</xsl:when>
<!-- otherwise just display the data as is-->
<xsl:otherwise>
```

```
<xsl:value-of select="."></xsl:value-of>
</xsl:otherwise>
</xsl:choose>

<!-- need a break if field is not empty -->
<xsl:variable name="empty_field">
<xsl:value-of select="."/>
</xsl:variable>
<xsl:if test="string-length($empty_field)!=0"><br></xsl:if>
</xsl:for-each>
</xsl:for-each>

<!-- Show separator line only if has one or more record -->
<xsl:variable name="row_data">
<xsl:value-of select="normalize-space(RS_DATA/ROW)"/>
</xsl:variable>
<xsl:if test="string-length($row_data)>0">
<xsl:text>- - - </xsl:text><br></xsl:if>
</xsl:if>

<!-- show More link only if there is next record set -->
<xsl:variable name="more_link">
<xsl:value-of select="normalize-space(RS_HEADER/METHOD[@NAME='GotoNextSet']/
@CAPTION)"/>
</xsl:variable>
<xsl:if test="string-length($more_link)>0">
<xsl:element name="A">
<xsl:attribute name="HREF">
<xsl:apply-templates select="RS_HEADER/METHOD[@NAME='GotoNextSet']"/>
</xsl:attribute>
<xsl:value-of select="$more_link"></xsl:value-of>
</xsl:element>
```

```
<br></br>
</xsl:if>
</xsl:template>
<!-- ===== Anchor URL Processing =====>
<!-- ANCHOR Template builds the URL for drilldowns and links -->
<xsl:template match="ANCHOR">
<xsl:text>start.swe?</xsl:text>
<xsl:apply-templates select="CMD|INFO"/>
</xsl:template>
<xsl:template match="CMD">
<xsl:value-of select="@NAME"/><xsl:value-of select="@VALUE"/>
<xsl:apply-templates select="ARG"/>
</xsl:template>
<xsl:template match="ARG">
<xsl:variable name="arg">
<xsl:if test="string-length(normalize-space()) >0">
<xsl:variable name="argstring">
<xsl:choose>
<xsl:when test="@NAME='Pu' or @NE='R' or @NAME='Rs'">
<xsl:value-of select="translate(normalize-space(),'%2B','+')"/>
</xsl:when>
<xsl:otherwise>
<xsl:value-of select="normalize-space()"/>
</xsl:otherwise>
</xsl:choose>
</xsl:variable>
<xsl:value-of select="$argstring"/>
</xsl:if>
</xsl:variable>
```

```
<xsl:text>&amp;</xsl:text>

<xsl:value-of select="@NAME"></xsl:value-of>=<xsl:value-of select="$arg"></xsl:value-of>

<!--<xsl:text>&#38;</xsl:text>-->

<!--<xsl:value-of select="@NAME"/>=<xsl:value-of select="translate($arg,'%2B','+')'"/>-->

</xsl:template>

<xsl:template match="INFO">

<xsl:variable name="info">

<xsl:if test="string-length(normalize-space(.)) >0">

<!--<xsl:value-of select="."/>-->

<xsl:value-of select="normalize-space(.)"/>

</xsl:if>

</xsl:variable>

<xsl:text>&amp;</xsl:text>

<xsl:value-of select="@NAME"/>=<xsl:value-of select="$info"/>

</xsl:template>

</xsl:stylesheet>
```

HTTP provides several means to allow Web Servers to obtain information from the browser. The most familiar example is when a user enters data into a form on a Web page and the data is sent to the Web Server, which can access the value of each form field. This example illustrates sending form field parameters to the Web Server with a POST method. In general, a browser can send cookies, headers, query string parameters, and form field parameters to the Web Server. Web Servers can also respond to the browser with cookies and custom headers. The Web Engine HTTP TXN Business Service (BS) allows Siebel eBusiness Applications to retrieve or set cookies, headers, and query string and form field parameters.

The Web Engine HTTP TXN Business Service can be invoked by scripts or by workflow. The inbound HTTP request to the Siebel Web Engine (SWE) is parsed and the BS returns property sets containing cookies, headers, or parameters. In addition, server variables, which are not a part of the HTTP request header, can also be retrieved. The BS can also set a custom cookie or header in the HTTP response header generated by the SWE. The BS gives complete control over the request header received and response header sent by the SWE.

## Scenario

A customer has a portal Web application that automatically links the user to Siebel Call Center with Siebel Web Single Sign On. The customer also wants to pass a custom header to the Siebel application during the single-sign-on process. The custom headers provide credentials to log into another application that is embedded in the Siebel application. Additionally, the portal Web application uses cookies to store a user profile that needs to be synchronized with the Siebel profile.

This example explains how to pass custom cookies, headers, and query string parameters to Siebel applications during login.

### Scenario

An external Web application such as ASP or JSP sets the header, cookie, or query string parameter and either sends the request to the Siebel application or redirects the browser to the Siebel application.

The following sample ASP code will log in a Siebel User and call the HTTP TXN Business Service to retrieve HTTP variables:

```
<% @language=VBScript %>

<% Response.buffer =true

'Set my URL parameter

'The following URL will log a Siebel User into Siebel and call the
HTTP TxN Business Service to retrieve HTTP variables.

strURL = "http://bng/callcenter/
start.swe?SWECmd=ExecuteLogin&SWEUserName=SADMIN&SWEPassword=SADMIN&myParm=Vanilla"

Set Http = Server.CreateObject ("WinHttp.WinHttpRequest.5")

Http.Open "GET", strURL, False

'Set my Cookie

Http.setRequestHeader "Cookie", "myCookie=Chocolate"

'Set my Header

Http.setRequestHeader "myHeader", "Strawberry"

Http.Send

Response.Write ("Request Sent... Check your Siebel Business Service
output")

Set Http = Nothing
```

%>

When the Siebel Web Engine receives the ExecuteLogin request, it triggers a user-defined runtime event to call a Business Service. Refer to *Personalization Administration Guide* for information on creating a runtime event and associating an action to the event. In this example, the action will call the HTTP TXN Business Service and invoke the getHTTPVars method.

The HTTP TXN Business Service is actually a wrapper function that calls the Web Engine HTTP TXN BS to retrieve HTTP and Server variables. The sample code below shows how to retrieve cookies, headers, and query string parameters and print them out to a text file. In an actual implementation, you may want to save the HTTP variables as profile attributes and use them in other scripts in the application.

## Sample HTTP TXN BS Code

```
Function Service_PreInvokeMethod (MethodName As String, Inputs As
PropertySet, Outputs As PropertySet) As Integer

If MethodName = "getHTTPVars" Then

Dim FS As Object

Dim oFile As Object

Dim oBS As Service

Dim In As PropertySet

Dim Out As PropertySet

Dim oChild As PropertySet

Set fs = CreateObject("Scripting.FileSystemObject")

If fs.FileExists("c:\testfile.txt") Then

Set oFile = fs.OpenTextFile("c:\testfile.txt", 8,0)

Else
```

```
Set oFile = fs.CreateTextFile("c:\testfile.txt", True)
End If

Set oBS = theApplication.GetService("Web Engine HTTP TXN")
Set In = theApplication.NewPropertySet()
Set Out = theApplication.NewPropertySet()

oFile.WriteLine("----Headers----")
oBS.InvokeMethod "GetAllRequestHeaders", In, Out
strName = Out.GetFirstProperty()
While strName <> ""
oFile.WriteLine(strName & " : " & Out.GetProperty(strName) )
strName = Out.GetNextProperty()
Wend

In.Reset
Out.Reset

oFile.WriteLine("----URL Params----")
oBS.InvokeMethod "GetAllRequestParameters", In, Out
strName = Out.GetFirstProperty()
While strName <> ""
oFile.WriteLine(strName & " : " & Out.GetProperty(strName) )
strName = Out.GetNextProperty()
Wend
```

```
In.Reset
Out.Reset

oFile.WriteLine("----Cookies----")
oBS.InvokeMethod "GetAllRequestCookies", In, Out
For nChild = 0 To Out.GetChildCount() - 1
Set oChild = Out.GetChild(nChild)
strName = oChild.GetFirstProperty()
oFile.WriteLine("TYPE : " & oChild.GetType())
oFile.WriteLine("VALUE : " & oChild.GetValue())
While strName <> ""
oFile.WriteLine(strName & " : " & oChild.GetProperty(strName) )
strName = oChild.GetNextProperty()
Wend
Next nChild

oFile.Close

Set oChild = Nothing
Set oFile = Nothing
Set fs = Nothing

Service_PreInvokeMethod = CancelOperation
```

### Scenario

```
TheApplication.GotoView "Home Page View (WCC)", Nothing
```

```
Else
```

```
Service_PreInvokeMethod = ContinueOperation
```

```
End If
```

```
End Function
```

## Sample Output by HTTP TxN BS

```
----Headers----  
COOKIE : myCookie=Chocolate  
HOST : bng  
CONTENT-LENGTH : 0  
CONNECTION : Keep-Alive  
USER-AGENT : Mozilla/4.0 (compatible; Win32;  
WinHttp.WinHttpRequest.5)  
MYHEADER : Strawberry  
ACCEPT : /*/*  
----URL Params----  
SWEUserName : SADMIN  
myParm : Vanilla  
SWECmd : ExecuteLogin  
SWEPassword : SADMIN  
----Cookies----  
TYPE : myCookie  
VALUE : Chocolate  
Domain :  
Path :  
Max-Age : -1
```

## Web Engine HTTP TXN Business Service API

Table 19 lists the methods exposed by the Web Engine HTTP TXN Business Service.

**Table 19. Web Engine HTTP TXN Business Service API**

<b>Method</b>	<b>Description</b>	<b>Parameters</b>
GetAllRequestCookies	Retrieves all request cookies sent from the client to the server.	<b>InputArguments:</b> Ignored. <b>OutputArguments:</b> Property Set hierarchy. Each cookie is a child Property Set with the TYPE property set to the cookie name.
GetAllRequestHeaders	Retrieves all request headers sent from the client to the server.	<b>InputArguments:</b> Ignored. <b>OutputArguments:</b> Property Set containing the HTTP Parameter name-value pairs.
GetAllResponseCookies	Retrieves all response cookies sent from the server to the client.	<b>InputArguments:</b> Ignored. <b>OutputArguments:</b> Property Set hierarchy. Each cookie is a child Property Set with the TYPE property set to the cookie name.
GetAllResponseHeaders	Retrieves all response headers sent from the server to the client.	<b>InputArguments:</b> Ignored. <b>OutputArguments:</b> Property Set containing the HTTP Header name-value pairs.
GetAllServerVariables	Retrieves all server variables.	<b>InputArguments:</b> Ignored. <b>OutputArguments:</b> Property Set containing the Server Variable name-value pairs.
GetClientCertificate	Retrieves the client certificate info.	<b>InputArguments:</b> Ignored. <b>OutputArguments:</b> Property Set containing certificate name-value pairs. Currently only returns Common Name (CN) property of the certificate.

**Table 19. Web Engine HTTP TXN Business Service API**

<b>Method</b>	<b>Description</b>	<b>Parameters</b>
GetRequestCookies	Retrieves the request cookies named in InputArguments.	<b>InputArguments:</b> Property Set containing the cookie names to retrieve. <b>OutputArguments:</b> Property Set hierarchy. Each cookie is a child Property Set with the TYPE property set to the cookie name.
GetRequestHeaders	Retrieves the request headers named in InputArguments.	<b>InputArguments:</b> Property Set containing the header names to retrieve. <b>OutputArguments:</b> Property Set containing the HTTP Header name-value pairs.
GetRequestInfo	Retrieves the request Web Session, Headers, Cookies, Parameters and Client Certificate information in one call.	<b>InputArguments:</b> Ignored <b>OutputArguments:</b> Property Set hierarchy. Each section is a child Property Set with the TYPE property set to 'Headers', 'Cookies', 'Parameters' or 'ClientCertificate'. The Web Session information is simply stored as properties of OutputArguments.
GetRequestParameters	Retrieves the request parameters named in InputArguments.	<b>InputArguments:</b> Property Set containing the parameter names to retrieve. <b>OutputArguments:</b> Property Set containing the HTTP Parameter name-value pairs.
GetResponseCookies	Retrieves the response cookies named in InputArguments.	<b>InputArguments:</b> Property Set containing the cookie names to retrieve. <b>OutputArguments:</b> Property Set hierarchy. Each cookie is a child Property Set with the TYPE property set to the cookie name.

**Table 19. Web Engine HTTP TXN Business Service API**

<b>Method</b>	<b>Description</b>	<b>Parameters</b>
GetResponseHeaders	Retrieves the response headers named in InputArguments.	<b>InputArguments:</b> Property Set containing the header names to retrieve. <b>OutputArguments:</b> Property Set containing the HTTP Header name-value pairs.
GetResponseInfo	Retrieves the response Headers and Cookies in one call.	<b>InputArguments:</b> Ignored. <b>OutputArguments:</b> Property Set hierarchy. Each section is a child Property Set with the TYPE property set to 'Headers' or 'Cookies'. Content Type and Status are simply stored as properties of OutputArguments.
GetServerVariables	Retrieves the server variables named in InputArguments.	<b>InputArguments:</b> Property Set containing the server variable names to retrieve. <b>OutputArguments:</b> Property Set containing the Server Variable name-value pairs.
GetWebSessionInfo	Retrieves the client's Web session information.	<b>InputArguments:</b> Ignored. <b>OutputArguments:</b> Property Set containing the Web session name-value pairs— SessionName; Cookie Name; SessionId; Web Session ID; SessionFrom (Value is 'URL' or 'COOKIE').

**Table 19. Web Engine HTTP TXN Business Service API**

Method	Description	Parameters
SetResponseCookies	Sets the response cookies to the values in InputArguments.	<p><b>InputArguments:</b> Property Set hierarchy. Each cookie is a child Property Set with the TYPE property set to the cookie name. The PERSISTENT property determines whether the cookie will persist between sessions. If the value is Y, the cookie persists between browser sessions. Otherwise, the cookie exists on a per session basis.</p> <p><b>OutputArguments:</b> Ignored.</p>
SetResponseHeaders	Sets the response headers to the values in InputArguments.	<p><b>InputArguments:</b> Property Set containing the HTTP Header name-value pairs.</p> <p><b>OutputArguments:</b> Ignored.</p>
SetResponseInfo	Sets the response Headers and Cookies in one call.	<p><b>InputArguments:</b> Property Set hierarchy. Each section is a child Property Set with the TYPE property set to 'Headers' or 'Cookies'. Content Type and Status are simply stored as properties of InputArguments.</p> <p><b>OutputArguments:</b> Ignored.</p>



# Index

## Symbols

- < APPLET > XML response tag, about and attributes 85
- < APPLICATION > XML response tag, about and attributes 84
- < COLUMN > XML response tag, about and attributes 86
- < FIELD > XML response tag, about and attributes 88
- < LIST > XML response tag, about and attributes 85
- < ROW > XML response tag, about and attributes 88
- < RS\_DATA > XML response tag, about 87
- < RS\_HEADER > XML response tag, about 86
- < SCREEN > XML response tag, about and attributes 84
- < VIEW > XML response tag, about and attributes 84

## Numerics

- 7.5 features 9

## A

- Accounts View, viewing in XML 54
- applet
  - external content, displaying outside 29
  - external content, displaying within 28
- architecture
  - Enterprise Application Integration, about 12
  - Portal Agents, about 13
  - XML Web interface 13
- ARG tag XML command block

- ARG parameter name-values pairs, table of 82
- attributes, table of 79
- description 78
- example 80
- required arguments 80
- authentication strategies, list of Portal Agents 16

## B

- business components, configuring to handle external data 27

## C

- CMD tag XML command block
  - attributes, table of 77
  - description 77
  - example 77
- content, integrating external
  - See *individual Portal Agent entries*

## D

- DeleteRecord command, about and example 101
- deleting
  - DeleteRecord, about and example 101
  - Execute Query, about and example 100
  - New Query, about and example 100
  - records, process of 99
- disposition types
  - list of 17
  - summary, table 22
- Document Type Definitions (DTD)
  - Inbound DTD 120
  - Outbound DTD 120

**E**

- EditField command, about and example 102
- EditRecord command, about and example 98
- EncodeURL command, about 46
- Enterprise Application Integration architecture, about 12
- errors
  - SWE log file, using to debug errors 46
  - XML response structure error, about contained in command block 83
- EXE tag XML command block
  - attributes, table of 76
  - description 76
  - example 76
- ExecuteLogin command, about and example 90
- ExecuteQuery command
  - deleting records, about and example 100
  - modifying records, about and example 97
  - querying items, about and example 94
- external content
  - applet, displaying outside 29
  - applet, displaying within 28
- external data, configuring business components to handle 27
- external host, defining 30

**F**

- features for 7.5 9
- Fixup Administration view, using to define a fixup type 38
- fixup type, defining 38
- Form Redirect disposition type, about and scenario 19

**G**

- GotoPageTab command
  - navigating to a screen, about and example 92

- picking records, about and example 102
- guide
  - 7.5 features 9
  - revision history 8

**H**

- high-interactivity applications, fixup type, about using for links 39
- history of revisions 8
- HTML attributes
  - IFrame command, about using to define 47
  - WebControl command, about using to define additional attributes 52

**I**

- IFrame command, about 47
- IFrame disposition type
  - about 18
  - summary, table 22
- Inbound DTD Document Type
  - Definitions 120
- Inline disposition type
  - about 17
  - restriction, use of 21
  - summary, table 22
- Internet Explorer
  - 4.0, Web control disposition type, about 19
  - 5.5, IFrame support, about 18
- InvokeMethod command, about and example 93

**L**

- log file, reviewing SWE log file 46
- login
  - credential, defining 40
  - page, reverse engineering 23
- login ID
  - Siebel login ID, about using UseSiebelLoginId 51

UserLoginId, about using to define for  
 Web application 50  
 Logoff command, about and example 92

## M

Mozilla browser, about 26

## N

NewQuery command  
 deleting records, about and example 100  
 modifying records, about and  
 example 97  
 querying items, example 94  
 NewRecord command, about and  
 example 95  
 NoCache command, about 47  
 NoFormFixup command, about 48

## O

Outbound DTD Document Type  
 Definition 120

## P

password  
 Siebel password, about using  
 UseSiebelLoginPassword  
 command 51  
 UserLoginPassword command, about  
 using 50  
 PickRecord command, about and  
 example 103  
 Portal Agent  
 See also *individual Portal Agent entries*  
 about and features 15  
 architecture, about 13  
 authentication strategies, list of 16  
 creating, overview of required tasks 23  
 data layer, about integrating data 16  
 disposition types summary, table of 22  
 disposition types, list of 17  
 Form Redirect disposition type, about and  
 scenario 19

IFrame disposition type, about 18  
 Inline disposition type, about 17  
 login requirements, determining 23  
 restrictions 21  
 Server Redirect disposition type, about  
 and arguments 20  
 SWE log file, reviewing 46  
 Symbolic URL commands, about 16  
 Web control disposition type 19  
 Portal Agent, administration  
 See also *individual Portal Agent entries*  
 content fixup, defining 38  
 external host, defining 30  
 Symbolic URL arguments, defining 35  
 Symbolic URL, defining 32  
 Web applications, defining 31  
 Portal Agent, command reference  
 See also *individual Portal Agent entries*  
 EncodeURL, about 46  
 IFrame, about 47  
 NoCache, about 47  
 NoFormFixup, about 48  
 PostRequest, about 49  
 PreLoadURL, about 49  
 UserLoginId, about 50  
 UserLoginPassword, about 50  
 UseSiebelLoginId, about 51  
 UseSiebelLoginPassword, about 51  
 WebControl, about 52  
 Portal Agent, configuring  
 See also *individual Portal Agent entries*  
 about 27  
 business components, configuring 27  
 external content, displaying outside an  
 applet 29  
 external content, displaying within an  
 applet 28  
 SWE log file, reviewing 46  
 Portal Agent, example  
 See also *individual Portal Agent entries*  
 external host, defining 42  
 login page, reviewing 41  
 step overview 40

- Symbolic URL arguments, defining 44
- Symbolic URL, defining 43
- test 45
- user login credentials, defining 45
- POST method, about using PostRequest to configure Portal Agent 49
- PostRequest command, about 49
- PreLoadURL command, about 49

## **Q**

- query string
  - Web Engine Interface, submitting requests as a query string 62
  - Web Server, submitting HTTP requests through 57
- XML request structure, constructing 74
- querying commands
  - ExecuteQuery command, about and example 94
  - NewQuery command, example 94

## **R**

- records, adding
  - NewRecord command, about and example 95
  - WriteRecord command, about and example 96
- records, deleting
  - DeleteRecord, about and example 101
  - ExecuteQuery, about and example 100
  - NewQuery, about and example 100
  - process of 99
- records, modifying
  - EditRecord command, about and example 98
  - ExecuteQuery command, about and example 97
  - NewQuery command, about and example 97
  - process of 96
  - WriteRecord command, about and example 99

- records, picking
  - EditField command, about and example 102
  - GotoPageTab command, about and example 102
  - PickRecord command, about and example 103
  - process of 101
  - WriteRecord command, about and example 103
- revision history 8

## **S**

- scenario
  - about 135
  - HTTP TXN BS, code sample 137
  - HTTP TxN BS, sample output 141
- screen
  - navigating to 92
  - navigating within 93
- Server Redirect disposition type
  - about and arguments 20
  - Symbolic URL argument, about using 37
- Session Management, about 15
- Session Proxy, about 16
- Session Re-Use, about 15
- Siebel login ID, about using
  - UseSiebelLoginId command 51
- Siebel Object Manager, Web server configuration and markup determination 55
- Siebel password, about using
  - UseSiebelLoginPassword command 51
- Siebel Web Engine (SWE)
  - See also *individual SWE entries*, and XML Web interface
  - HTML output, about configuring for 55
- Siebel Wireless WML, about setting
  - Wireless parameter 120
- Siebel XML
  - See also XML
  - accessing, about 54

- manipulating with stylesheets and XSLT 127
- XML-specific template tag, about and example 127
- Simple Portal Agents, about authentication strategy 16
- Single Sign-On Portal Agents authentication strategy, about 16
- Single Sign-On technology (SSO), about 15
- SSO Systems Administration view, using to specify Web application 40
- stylesheets, defining SWT stylesheet tags 127
- SWE API
  - SWE arguments, table of 116
  - SWE commands, table of 105
  - SWE methods, table of 110
- SWE arguments, table of 116
- SWE commands
  - SWEAC command, using to string commands together 105
  - table of 105
- SWE log file, reviewing 46
- SWE methods, table of 110
- SWEAC command, using to string commands together 105
- Symbolic URL
  - See also* Portal Agent, example arguments, defining 35
  - business component, configuring 27
  - commands, about 16
  - defining 32
  - disposition types, list of 17
  - EncodeURL, about using to specify encoding arguments 46
  - Inline disposition type 17
  - multiple disposition types, about 15
  - PreLoad URL, about using 49
  - Server Redirect disposition type, about using 37

**T**

- Time-out Handling, about 15

**U**

- UserLoginId command, about 50
- UserLoginPassword command, about 50
- UseSiebelLoginId command, about 51
- UseSiebelLoginPassword command, about 51

**W**

- Web application
  - defining 31
  - specifying and defining login credentials 40
- Web control disposition type
  - about 19
  - summary, table 22
- Web Engine HTTP TXN Business Service
  - about invoking 135
  - HTTP TXN BS, code sample 137
  - HTTP TxN BS, sample output 141
  - methods, table of 142
  - scenario 135
- Web Engine Interface
  - submit requests, about using to 60
  - submit requests, as a query string 62
  - submit requests, as an XML command block 63
- Web Server
  - query string, using to send HTTP requests 57
  - XML command block, using to send HTTP requests 59
- WebControl command, about 52
- WriteRecord command
  - adding records, about and example 96
  - modifying records, about and example 99
  - picking records, about and example 103

**X**

- XML
  - See also* Siebel XML
  - HTTP response, WML response 120

- HTTP response, XML response tags (table) 84
- markup determination, process steps 55
- Siebel Wireless WML, about setting Wireless parameter 120
- XML-specific template tag, about and example 127, 128
- XML command block
  - ARG tag 78
  - CMD tag 77
  - EXE tag 76
  - Web Engine Interface, submitting requests 63
  - Web Server, using to send HTTP requests 59
  - XML tags, table of 74
- XML commands
  - deleting records 99
  - ExecuteLogin command, about and example 90
  - ExecuteQuery command, about and example 94
  - GotoPageTab command, about and example 92
  - InvokeMethod command, about and example 93
  - Logoff command, about and example 92
  - modifying records 96
  - New Query command, example 94
  - NewRecord command, example 95
  - objects available on screen, viewing 90
  - picking records 101
  - WriteRecord command, example 96
- XML request structure
  - query string, constructing 74
  - XML command block, constructing 74
- XML response structure
  - about 83
  - error, about contained in command block 83
  - XML response tags, about and table of 83
- XML response tag
  - HTML response, about 90
  - response syntax format (example) 88
  - table of tags, description, and attributes 84
- XML Web interface
  - Accounts View, viewing in 54
  - architecture, about 13
  - overview 53
  - Siebel XML, about accessing 54
- XML Web interface, connecting to query string, using to send HTTP requests 57
  - Web Engine Interface, about using to submit requests 60
  - Web Engine Interface, submitting requests as a query string 62
  - Web Engine Interface, submitting requests as an WML command block 63
  - XML command block, using to send HTTP requests 59
- XSL stylesheets, defining tags 127