Siebel Portal Framework Guide

Siebel Innovation Pack 2013
Version 8.1/8.2, Rev. A
December 2013

ORACLE

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Contents

Chapter 4: Delivering Content to External Web Applications

1 What's New in This Release

What's New in Siebel Portal Framework Guide, Version 8.1 / 8.2, Rev. A

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

Table 1. What's New in Siebel Portal Framework Guide, Version 8.1/8.2, Rev. A

Topic	Description
"SWE Commands Available in Siebel Open UI" on page 79	Added a table listing the SWE commands that are available in Siebel Open UI. Some of the functionality described in this guide does not apply to the Siebel Open UI client.

What's New in Siebel Portal Framework Guide, Version 8.1, Rev. A and Version 8.2, Rev. A

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

Table 2. What's New in Siebel Portal Framework Guide, Version 8.1, Rev. A and Version 8.2, Rev. A

Topic	Description
"About Portal Agents" on page 11	Session time-out functionality is available only in standard interactivity applications.
"FreePopup Command" on page 35	Added a description of the FreePopup command.

2 Siebel Portal Framework

This chapter provides an overview of Oracle's Siebel Portal Framework and summarizes the technologies that make up the Portal Framework. It contains the following information:

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 CRM data to external applications.
- Integrate external application business logic and data with Siebel Business 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 Enterprise Application Integration* and other EAI titles on the *Siebel Bookshelf*. The *Siebel Bookshelf* is available on Oracle Technology Network (OTN) and Oracle Software Delivery Cloud. It might also be installed locally on your intranet or on a network location.

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. For more information about Portal Agents, see "About Portal Agents" on page 11.

XML Web Interface

In enterprises where a non-Siebel portal framework is already established, you might have to be able to deliver Siebel content to other applications and frameworks.

If you have configured your Siebel application to display data in a high interactivity client, then the XML Web interface provides you with a mechanism to deliver Siebel data to external applications as XML documents. For more information, see Chapter 4, "Delivering Content to External Web Applications."

NOTE: The Siebel Open UI client supports HTML markup only. If you have configured your Siebel application to display data in a Siebel Open UI client, then you must use a different technology to send this content. For information about how to send this content from Siebel Open UI, see *Configuring Siebel Open UI*. The SWE API described in "SWE API" on page 74 includes several SWE commands that are available in Siebel Open UI.

Both methods provide the external application with a flexible format for integrating Siebel data into its user interface.

3 Integrating External Content

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

About 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 application Web page.

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

- **Single sign-on technology (SS0).** 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 user name and password, more than once for each work session. For more information about single sign on, see *Siebel Security Guide*.
- **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. For more information, see "About Disposition Types" on page 12.

- **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 Disposition Type" on page 13.
- **Symbolic URL commands.** Commands that direct the Portal Agent to assemble the URL for the external application in several ways. These include dynamically referencing the user's user name and password, retrieving stored user name 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 34.

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. For more information about Siebel EAI, see *Overview: Siebel Enterprise Application Integration*.

This topic contains the following information:

Portal Agents and Authentication Strategies

Portal Agents can be configured to support different authentication strategies:

- **Simple Portal Agents.** The external application does not require any authentication parameters.
- **Single Sign-On Portal Agents.** The external application requires authentication parameters. Form-based Portal Agents send authentication parameters as part of the body portion of the HTTP request.

For more information about authentication, see *Siebel Security Guide*.

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 following:

- 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 which one suits your integration needs. Each disposition type is discussed in one of the following topics:

- "Form Redirect Disposition Type" on page 14

For information about defining symbolic URLs, see "Defining Symbolic URLs" on page 23.

Inline Disposition Type

With a symbolic URL disposition type of Inline, theSiebel Server receives content sent by anexternal 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 in such a way that it appears to the user as one integrated application rather than from different application servers.

The Inline disposition type supports session management. The Siebel Server uses session management to manage session cookies and automatically log in again to an external application after a time out occurs.

The Inline disposition type requires that:

- The page you are integrating does not include complex JavaScript and does not reference frames.
- The maximum number of characters in the calling URL is 2048.
- No methods other than the `GET` method are invoked.

If the Inline disposition type is not appropriate, then you might try the IFrame disposition type.

IFrame Disposition Type

Use this disposition type when aspects of the external application do not allow content to be aggregated with other Siebel content. For more information, see "Portal Agent Restrictions" on page 15.

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 topopulate 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 a method that works.

The IFrame disposition type supports JavaScript and frames. Therefore, if the Inline disposition type does not work, then 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 works 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 that you are trying to integrate refer to top-level JavaScript objects, then you might use the Web Control disposition type instead, if it is applicable.

Contextual Navigation Between Siebel Business Applications and Oracle Business Intelligence Pages

When an Oracle® Business Intelligence (Oracle BI) page is integrated with a Siebel application through the portal framework and the portal content is dependent on the Siebel record, any change or update of the record in the Siebel application must also be reflected in the portal content. For example, for an Oracle BI applet embedded in a view with the Account List applet, its content dynamically changes at the same time that the content is changed within the Account List applet. To enable this behavior, you must do the following:

- Define a symbolic URL. For more information, see "Defining Symbolic URL Arguments" on page 25.
- Set parameters for the symbolic URL. For more information, see "Portal Agent Command Reference" on page 34.

Web Control Disposition Type

For the high interactivity client only, you can use the Web Control disposition type when the 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. For more information, see "Portal Agent Restrictions" on page 15.

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 not available in Siebel Open UI.

Form Redirect Disposition Type

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 type isusually displayed in a new window, rather than inline with other Siebel applets.

The Form Redirect disposition type is not commonly used with Siebel Business Applications.

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 might not be amenable to inline aggregation methods.
- Specific frame references in the returned content referring to global frames (_NEW, _TOP, .parent()) might not be amenable to inline aggregation methods.
- Reliance on JavaScript functions defined in (assumed) external frames might not be amenable to inline aggregation methods.
- URLs that are created dynamically by JavaScript might not be amenable to any fixup techniques, because the URLs would not be easily parsed on the HTML content.

For these reasons, an Inline disposition type does not work often. However, if you control both the Siebel application instance and the external application, and can resolve some of these issues, then the Inline disposition type might work correctly. For more information about the Inline disposition type, see "Inline Disposition Type" on page 13.

If you do not have control over the external application, the IFrame disposition type is the method most likely to 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 Disposition Type" on page 13.

Disposition Types Summary

Table 3 summarizes the characteristics of each disposition type.

Table 3. Disposition Types Summary

Disposition Type	Benefits	Limitations
Inline	■ Inline integration with the Siebel user interface. ■ Session management, including managing session cookies and automatic re-login after time out. ■ Opens an external URL in a new popup window.	■ Only works in very few cases. ■ Does not work with complex JavaScript. ■ Does not work if there are reference to frames. ■ Supports the GET method only. ■ URL limited to 2048 characters.
IFrame	■ Inline integration with the Siebel user interface. Supports complex JavaScript. ■ Supports references to frames. ■ Session Keep Alive supported. ■ Works for most cases.	■ No session management. ■ Does not support frames that reference top-level JavaScript objects. ■ Does not open an external URL in a popup window.
Web Control	Supports frames that reference top-level JavaScript Objects, because JavaScript does not refer to objects outside of the Web control.	■ For the high interactivity client only. ■ No session management. ■ Browser functionality, such as the back button, is only available by right-clicking in the Web control. ■ ActiveX objects that contain other objects are reset if you change tabs and then return to the Web control. ■ Does not open an external URL in a popup window. ■ Web Control requires more system overhead than IFrame.

Process of Creating Portal Agents

To create a Portal Agent, perform the following tasks:

1. "Determining the Login Requirements" on page 17.
2. "Configuring Business Components to Handle External Data" on page 19.
3. Complete one of the following:
   - "Displaying External Content Within an Applet" on page 20.
   - "Displaying External Content Outside of an Applet" on page 20.
4. "Defining Web Applications" on page 22.
5. "Defining Symbolic URLs" on page 23.
6. "Defining Symbolic URL Arguments" on page 25.

Determining the Login Requirements

Before you configure Portal Agents, you must 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 might process login field values prior to delivering the POST back to the application server, session values might be encoded in the form itself, or session values might be stored in the browser's session cookies.

This task is a step in "Process of Creating Portal Agents" on page 17.

To reverse-engineer a login page

1. Navigate to the external application's login page and determine whether the external application uses authentication.

   For more information, see "Defining Symbolic URLs" on page 23.

2. If the external application uses form-based authentication, then 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 (the form might 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 that are not part of the <input> tags.

4 Determine whether the method attribute of the <form> tag is POST.

If it is POST, then you must define the PostRequest command as an argument of the symbolic URL. For more information, see "Defining Symbolic URL Arguments" on page 25 and "PostRequest Command" on page 38.

If it is GET, then you do not have to define a symbolicURL 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 (/), then use this URL as the base of the Portal Agent.

If it is a relative address, then you also have 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. For more information, see "Defining the External Host" on page 21 and "Defining Symbolic URLs" on page 23.

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 before it times out. In this case, it might not be possible to configure a Portal Agent. Define any argument values contained in the target URL as symbolic URL arguments. For more information, see "Defining Symbolic URL Arguments" on page 25.

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 might 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), then you can use UserLoginId Command and UserLoginPassword Command commands to instruct the Portal Agent to retrieve the user's credentials from the user's My Logins view. For more information, see "Defining End-User Login Credentials" on page 29.

- If there are hidden fields with no values, then, 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, then the Portal Agent does not send this request to the target application server, because there is no value to put in its place.

You define the input fields and values as symbolic URL arguments. For more information, see "Defining Symbolic URL Arguments" on page 25.

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 topic 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 might be required to perform. For example, after you configure an applet to display external content, you might 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 Business Applications and are outside the scope of this book. For more information about configuring Siebel Business Applications, see *Configuring Siebel Business Applications*.

This topic contains the following information:

Configuring Business Components to Handle External Data

To configure business components to handle external data using a symbolic URL, you must create a new calculated field on the business component. Rather than representing structured content, such as records in a database, this field represents 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.

This task is a step in "Process of Creating Portal Agents" on page 17.

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 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 20.

This task is a step in "Process of Creating Portal Agents" on page 17.

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 19.
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 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 content inside an applet, by using an Applet Control or List Column. For more information, see "Displaying External Content Within an Applet" on page 20.

This task is a step in "Process of Creating Portal Agents" on page 17.

To display content outside of an applet

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

Name	Value
FieldRetrievalType	Symbolic URL
SymbolicURL	*[name of symbolic URL]*

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 Administration - Integration screen in the Siebel Web client. As described in the following topics, these views allow you to define how to handle links, define the external host, and define the HTTP request that is sent to the external host.

This topic contains the following information:

Defining the External Host

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

- Maintain external host names in a single place.
- Define how to handle (fix) links after external HTML content is rendered.

To define a data host

1. Navigate to the Administration - Integration screen, and then WI Symbolic URL List.
2. From the drop-down menu, select Host Administration.
3. 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	Leave this value blank. For more information, see "Defining Symbolic URLs" on page 23.

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 specify whether they share the same session.

There might be cases in which you do not want requests to share the same session. For example, you might 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 it shares 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.

This task is a step in "Process of Creating Portal Agents" on page 17.

To define a Web application

1. Navigate to the Administration - Integration screen, and then WI Symbolic URL List.
2. From the drop-down menu, select Web Application Administration.
3. Enter a record and complete the fields.

   Some of the fields are described in the following table:

Field	Description
Shared	Indicates whether 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 to construct the HTTP request to the external application and to define any arguments and values to be sent as part of the request.

This task is a step in "Process of Creating Portal Agents" on page 17.

To define a symbolic URL

1. Navigate to the Administration - Integration screen, and then WI Symbolic URL List.
2. From the drop-down menu, select Symbolic URL Administration.
3. In the Symbolic URL Administration list view, enter a new record.

   Some of the fields are defined in the following table:

Field	Description
URL	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 might have many symbolic URLs pointing to one host. If the host name changes, then you only need to change it in the Host Administration applet rather than having to change it in several symbolic URL definitions. For example, `https://Virtual_Host/path...` **NOTE:** Use the Secure Sockets Layer protocol (SSL) with symbolic URLs to ensure that communication is secure. For more information about using SSL, see *Siebel Security Guide*. For applications that use form-base authentication, the URL is identifiedby the action attribute of the `<form>` tag. For more information, see "Determining the Login Requirements" on page 17.
Host Name	The Virtual Name of the host defined in the Host Administration view.

Field	Description
Fixup Name	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: ■ **Default.** 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. ■ **InsideApplet.** 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. ■ **OutsideApplication.** This fixup converts all of the relative links to absolute links using the host and path of the parent URL. No links are proxied.
Multivalue Treatment	Determines how arguments are handled. Possible values are: ■ **Comma Separated.** 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. ■ **Separate Arguments.** Instructs SWE to enter separate arguments for each value defined in the two Argument Value fields. ■ **Use First Record Only.** Uses the first record in the current record set.

Field	Description
SSO Disposition	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: ■ **Inline.** Proxies the request through the Siebel Server and displays content inline with other applets on a view. ■ **IFrame.** Uses the <iframe> tag to display content inline with other applets on a view. ■ **Web Control.** Uses an ActiveX control to display content inline with other applets on a view. Browsers displaying symbolic URLs of type Web Control must be set to handle ActiveX controls. For more information about browser security settings, see *Siebel Security Guide*. ■ **Form Redirect.** 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. ■ **Server Redirect.** 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. For more information, see "Portal Agent Restrictions" on page 15. For detailed descriptions of each disposition type, see "About Portal Agents" on page 11.
Web Application Name	Associates a Web application with this symbolic URL. For more information about Web applications, see "Defining Web Applications" on page 22.

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 an 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.

For information about how to determine required data for applications that use form-based authentication, see "Determining the Login Requirements" on page 17.

Symbolic URL arguments also allow you to implement commands which you use to define the behavior of Portal Agents. For usage descriptions of available commands, see "Portal Agent Command Reference" on page 34.

This task is a step in "Process of Creating Portal Agents" on page 17.

To define symbolic URL arguments

1. Navigate to the Administration - Integration screen, and then WI Symbolic URL List.
2. From the drop-down menu, select Symbolic URL Administration.
3. In the Symbolic URL Administration list view, select the symbolic URL that you want to configure.
4. In the Symbolic URL Arguments form, 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	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 an attribute-value pair appended to the URL. For argument types of commands, the Name can usually be anything. The only exception to this is for the EncodeURL and PreloadURL commands. For more information, see "Portal Agent Command Reference" on page 34.
Required Argument	When this field is checked (default) the argument must have a value. If you are configuring an argument that does not have a value, then uncheck the Required field. If an argument has no value and the Required field is checked, then the request is not sent because there is no value to append to the URL.
Argument Type	The Argument Type determines the source of the data to be send along in the HTTP request. Possible values are: ■ **Constant.** Sends the value defined in the Argument Value field in the request. ■ **Field.** Sends the value of a single-value or multi-value field from the current Siebel business component. ■ **Profile Attribute.** Sends the value of a field from the Siebel Personalization business component. ■ **URL Argument.** Data comes from the named argument of the current request. ■ **Language Value.** The user's current language setting; for example, ENU. ■ **Command.** Implements commands that allow you to affect the behavior of the symbolic URL. For a complete list of commands, see "Portal Agent Command Reference" on page 34.

Field	Description
Argument Value	The value of the argument varies depending on the Argument Type. Descriptions of possible values for each argument type are described below. If the Argument Type is one of the following: ■ **Constant.** The Argument Value is the second part of the attribute-value pair that is appended to the URL. ■ **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. ■ **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 ■ **URL Argument.** The Argument Value defines the name of the argument on the incoming SWE request. ■ **Language Value.** The Argument Value is left null. ■ **Command.** The Argument Value typically defines the name of the command. For more information, see "Portal Agent Command Reference" on page 34.
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 notchecked, then the value is 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.

Configuring Multiple Symbolic URLs and Hosts for Alternative Execution Locations

You can configure multiple symbolic URLs and symbolic URL hosts, to execute applications in alternative locations (for example, for testing or demonstration purposes). This topic contains the following information:

■ "Configuring Alternative Symbolic URLs" on page 28

■ "Configuring Alternative Symbolic URL Hosts" on page 28

NOTE: When you use an alternative symbolic URL host, all symbolic URLs in the application that are configured to use that host will use the alternative host name. In contrast, when you use alternative symbolic URLs, each symbolic URL used in the application must have its own alternative symbolic URL. Therefore, you can reduce the effort required to execute the application in an alternative location by using an alternative symbolic URL host rather than a symbolic URL.

Configuring Alternative Symbolic URLs

To use an alternative symbolic URL, define the additional symbolic URL at the Symbolic URL Administration view, and specify the following parameter in the [DataSources] section of the application's configuration file:

SymbolicURLSuffix. The value of this parameter is appended to the end of the name of the default symbolic URL to specify the name of the alternative symbolic URL.

For example, if the parameter SymbolicURLSuffix is set to _MyDemo in the application's configuration file, and the default symbolic URL name is AccountNews, then the symbolic URL that is used when the application is executed is AccountNews_MyDemo. The URL value associated with the AccountNews_MyDemo symbolic URL in the Symbolic URL Administration page is used.

NOTE: When you define the alternative symbolic URL, its name must match the name of the existing symbolic URL with the value of the SymbolicURLSuffix parameter appended to it.

For more information about defining symbolic URLs, see "Defining Symbolic URLs" on page 23.

Configuring Alternative Symbolic URL Hosts

To use an alternative symbolic URL host, define the additional symbolic URL host at the Host Administration view, and specify the following parameter in the [DataSources] section of the application's configuration file:

SymbolicURLHostSuffix. This value is appended to the end of the name of the existing symbolic URL host to specify the name of the alternative symbolic URL host.

For example, if the parameter SymbolicURLHostSuffix is set to _demo in the application's configuration file, and the existing host name is ABC, then the new host name is ABC_demo. The host name value associated with ABC_demo in the Host Administration page is used.

NOTE: When you define the alternative symbolic URL host, its name must match the name of the existing symbolic URL host with the value of the SymbolicURLHostSuffix parameter appended to it.

For more information about defining hosts, see "Defining the External Host" on page 21.

Defining Content Fixup

The Fixup Administration view allows you to define how links embedded within external HTML content are rendered in the Siebel user interface. The fixuptypes that you define here will be associated with symbolic URLs.

To define a fixup type

1. Navigate to the Administration - Integration screen, and then WI Symbolic URL List.
2. From the drop-down menu, select Fixup Administration.

3 Enter a new record and define the fields.

Some of the fields are described in the following table:

Field	Comments
Link Context	Select one of the following values: ■ **Do Nothing.** This fixup does not affect anyof the links. The links (relative or absolute) remain as they are with the content being passed back in its original form. ■ **Outside Application.** This fixup converts all of the relative links to absolute links using the host and path of the parent URL. No links are proxied. ■ **Inside Application.** 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. ■ **Inside Applet.** 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.
Context View Name	Name of the view that displays 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 *AnyName* 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 the Administration - Integration screen, and then SSO Systems Admin List.

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 UserLoginId Command and UserLoginPassword Command 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, then, in the SSO System Users list, enter end-user login names and passwords.

Example Portal Agent

This topic provides an example of using a symbolic URL to integrate content from an external site. The high-level steps to do this are:

1 "Review the Login Form" on page 30.

2 "Define the External Host" on page 31.

3 "Define the Symbolic URL" on page 32.

4 "Define Symbolic URL Arguments" on page 33.

5 "Define User Login Credentials" on page 33.

6 "Testing the Integration" on page 34.

Each of these steps is described in the topics that follow. This example uses www.example.com, which does not have the login page and other elements described here; substitute your actual site.

NOTE: This example assumes that the underlying objects are already configured to support the symbolic URL. For more information, see "Portal Agent Configuration" on page 19.

Review the Login Form

By reviewing the login page at www.example.com, you can determine the target URL of the Action attribute and the required arguments that are being passed to the Web application. Assume that www.example.com has a login page that contains the following <form> 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/> <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.example.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 17.

Define the External Host

The external host is simply the address of the login page. In this example, it is www.example.com. Be sure to provide a meaningful name in the Virtual Host Name field. This value is used instead of the actual host name when you define the symbolic URL. This makes administration easier if the host name changes. Also notice that there is no value for the Authentication Type.

Figure 1 shows the external host defined for this example.

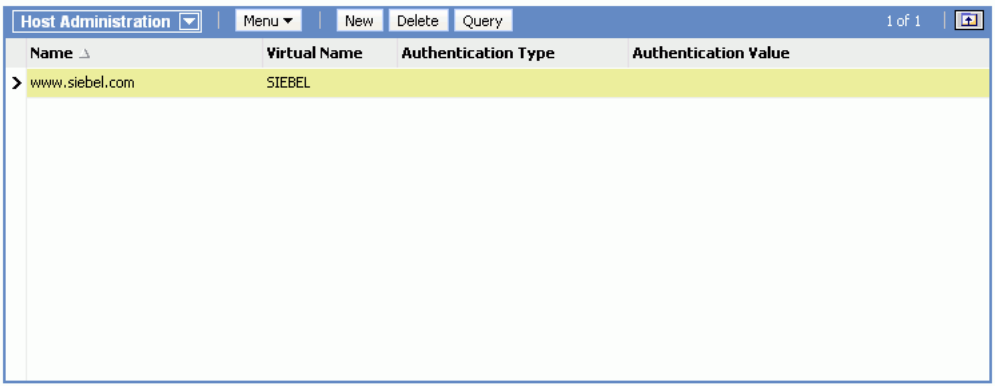

Figure 1. External Host Administration

For more information, see "Defining the External Host" on page 21.

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 is displayed in the browser using the <iframe> tag and therefore, it is recommended that links not be fixed up in any way.

Figure 2 shows the symbolic URL defined for this example.

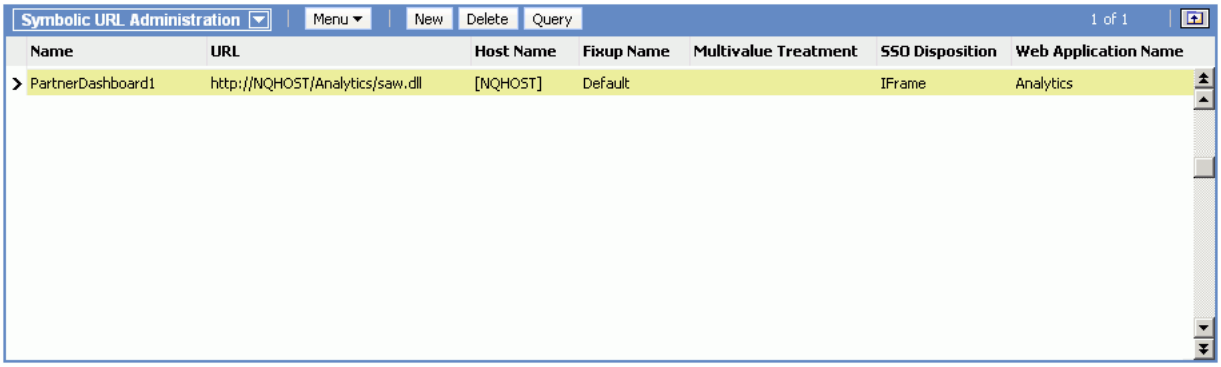

Figure 2. Symbolic URL

For more information about defining symbolic URLs, see "Defining Symbolic URLs" on page 23.

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 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 that 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.

Symbolic URL Arguments | Menu | New | Delete | Query | 1 - 7 of 7

Name	Required Argument	Argument Type	Argument Value	Append as Argument	Substitute in Text	Sequence #
IFrameLogin:Cmd	✓	Constant	Logon	✓		1
Cmd	✓	Constant	PortalPages	✓		2
IFrameLogin:nqUser	✓	Command	UseSiebelLoginId	✓		3
IFrameLogin:nqPassword	✓	Command	UseSiebelLoginPassword	✓		4
PortalPath	✓	Constant	/shared/Partner/_Portal/Channel Executive	✓		5
PostRequest	✓	Command	PostRequest	✓		6
IFrameLogin:Syndicate	✓	Constant	Siebel	✓		7

Figure 3. Symbolic URL Arguments

For more information about symbolic URL arguments, see "Defining Symbolic URL Arguments" on page 25. For more information about symbolic URL commands, see "Portal Agent Command Reference" on page 34.

Define User Login Credentials

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

- The user name is Joe_Smith@example.com.
- The password is abracadabra.

Testing the Integration

After completing the previous steps, you can test the integration.

To test the integration

1. Log out of the application.
2. Log back in as the test user.
3. Navigate to the applet or Web page item that is associated with the symbolic URL.

   Content from the external host, in this case example.com, is displayed in the Siebel user interface.

Reviewing the SWE Log File

Reviewing the SWE log file can help you to debug errors in your Portal Agent configuration.

- The location of the log file is *SIEBSRVR_ROOT*\log.
- The name of the log files are swelog_*pid*.txt and sweusage_*pid*.txt, where *pid* is the process ID of the corresponding Siebel process.

For more information about log files and about configuring log levels, see *Siebel System Monitoring and Diagnostics Guide*.

Portal Agent Command Reference

Portal Agent commands allow you to carry out actions 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. For more information, see "Defining Symbolic URLs" on page 23.

The following commands are described in this topic:

- "UseSiebelLoginId Command" on page 39
- "UseSiebelLoginPassword Command" on page 39
- "WebControl Command" on page 39

EncodeURL Command

Use the EncodeURL command to specify whether to encode arguments appended to the symbolic 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.

Define the fields in the Symbolic Arguments applet. See Table 4.

Table 4. Symbolic URL Arguments

Field	Value
Name	EncodeURL
Argument Value	*TRUE* or *FALSE*

FreePopup Command

Use the FreePopup command to show portal contents in a popup window.

The high interactivity and standard interactivity modes are implemented differently for FreePopup.

- For high interactivity, when the symbolic URL contains the FreePopup command, it notifies the client that the popup is a free one and the client displays the contents in the popup window.
- For standard interactivity, the client opens a new window from the first popup and closes the old one. The second popup is a free popup.

FreePopup is supported for FormRedirect, the only disposition type available for opening a portlet in a popup.

To start the external application as a full browser window, use the values in Table 5.

Table 5. Symbolic URL Arguments

Name	Required Argument	Argument Type	Argument Value	Sequence	Append as Argument
FreePopup	True	Command	True	1	True
FullWindow	True	Command	True	2	True

To start the external application as a modal window, use the values in Table 6.

Table 6. Symbolic URL Arguments

Name	Required Argument	Argument Type	Argument Value	Sequence	Append as Argument
PopupSize	True	Command	750x500	1	True
FreePopup	True	Command	True	2	True

IFrame Command

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

Define the fields in the Symbolic URL Arguments applet. See Table 7.

Table 7. Symbolic URL Arguments

Field	Value	Example
Name	Any Name	None
Argument Value	IFrame [*attribute*]=[*value*]	IFrame Height=100 Width=500

Disposition Types

Use the IFrame command with the IFrame disposition type.

IsRecordSensitive Command

Use the IsRecordSensitive command to turn on or off the record-sensitive feature. Set the value to TRUE to ensure that a child applet with a symbolic URL is refreshed on the parent record, for instance, when you embed an Analytics report as a child applet with a requirement that it display contextual information.

This command is turned off by default. Set this argument value to TRUE in the Symbolic URL Arguments configuration.

Define the fields in the Symbolic URL Arguments applet. See Table 8.

Table 8. Symbolic URL Arguments

Field	Value
Name	IsRecordSensitive
Argument Value	TRUE

NoCache Command

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.

Define the fields in the Symbolic URL Arguments applet. See Table 9.

Table 9. Symbolic URL Arguments

Field	Value
Name	Any name
Argument Value	NoCache

NoFormFixup Command

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.

Define the fields in the Symbolic URL Arguments applet. See Table 10.

Table 10. Symbolic URL Arguments

Field	Value
Name	Any name
Argument Value	NoFormFixup

PreLoadURL Command

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.

Define the fields in the Symbolic URL Arguments applet. See Table 11.

Table 11. Symbolic URL Arguments

Field	Value
Name	PreLoadURL
Argument Value	[*URL*]

PostRequest Command

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.

Define the fields in the Symbolic URL Arguments applet. See Table 12.

Table 12. Symbolic URL Arguments

Field	Value
Name	Any Name
Argument Value	PostRequest

UserLoginId Command

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.

For more information about how user login IDs are entered into this business component, see "Defining End-User Login Credentials" on page 29.

Define the fields in the Symbolic URL Arguments applet. See Table 13.

Table 13. Symbolic URL Arguments

Field	Value
Name	[*input field name*]
Argument Value	UserLoginId

UserLoginPassword Command

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.

For more information about how user passwords are entered into this business component, see "Defining End-User Login Credentials" on page 29.

Define the fields in the Symbolic URL Arguments applet. See Table 14.

Table 14. Symbolic URL Arguments

Field	Value
Name	[*input field name*]
Argument Value	UserLoginPassword

UseSiebelLoginId Command

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

Define the fields in the Symbolic URL Arguments applet. See Table 15.

Table 15. Symbolic URL Arguments

Field	Value
Name	[input field name]
Argument Value	UseSiebelLoginId

UseSiebelLoginPassword Command

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

Define the fields in the Symbolic URL Arguments applet. See Table 16.

Table 16. Symbolic URL Arguments

Field	Value
Name	[*input field name*]
Argument Value	UseSiebelLoginPassword

WebControl Command

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

Define the fields in the Symbolic URL Arguments applet. See Table 17.

Table 17. Symbolic URL Arguments

Field	Value	Example
Name	Any Name	None
Argument Value	WebControl [*attribute*]=[*value*]	WebControl Height=100 Width=500

4 Delivering Content to External Web Applications

This chapter describes how to use the XML Web Interface to deliver content to external portal frameworks and Web application environments when you configure your Siebel application to display data in a high interactivity client. This chapter contains the following information:

Overview of the XML Web Interface

You can use the XML Web Interface to deliver content to external portal frameworks and Web application environments when you configure your Siebel application to display data in a high interactivity client.

NOTE: The Siebel Open UI client supports HTML markup only. If you configure your Siebel application to display data in a Siebel Open UI client, then you must use a different technology to send this content. For information about how to send this content from Siebel Open UI, see *Configuring Siebel Open UI*. The SWE API described in "SWE API" on page 74 includes several SWE commands that are available in Siebel Open UI.

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. Some specialized applets might have limited support for the XML interface.

Developers can configure Siebel Business 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.

The XML interface provides access to Siebel Business Applications through the Siebel Web Engine (SWE). SWE generates the user interface in HTML or WML, using views, applets, and Siebel Web 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.

NOTE: Most Siebel applets, with the exception of applets based on 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

Accessing Siebel XML

By default, Siebel Business Applications present a standard HTML-based user interface (UI) to end users. When you use the XML interface, the standard architecture changes slightly; an XMLinterface layer is introduced. The XML interface layer accesses Siebel Business Applications through the SWE using the UI constructs, views, applets, and Siebel Web templates. It provides visibility into Siebel business objects and business components. TheseUI constructs provide not only filtering andaccess 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 Business Applications 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.

NOTE: The Siebel Open UI client supports HTML markup only. For more information, see "Overview of the XML Web Interface" on page 41.

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.

Siebel Object Manager 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 is set using the SWEMarkup parameter for the Application Object Manager. Based on browser or device detection or parameters set on the inbound request, this default markup might be overridden.

NOTE: The Siebel Open UI client supports HTML markup only. For more information, see "Overview of the XML Web Interface" on page 41.

The following is a summary of how the markup is 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 **SWESetMarkup parameter.** 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 takes information from the request header and look up the designated markup in the device table. The resulting markup is passed to the next step. If no match is found in the device table, then 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 is generated. The server default markup is designated by the SWEMarkup parameter the Application Object Manager.

Table 18 shows a summary of the markup that is generated for a given request based on the intersection of the server configuration markup and the markup from the user agent service.

Table 18. 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 you use the XML interface in conjunction with the Siebel Wireless WML-based application, the Wireless parameter must be set to TRUE (the default value) for the Application Object Manager for the Siebel Wireless application.

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

Connecting to the XML Web Interface

The XML Web Interface can be used against any Siebel Business Applications. Requests to generate XML from a Siebel application can be submittedthrough a Siebel Web Server using a query string or an XML command block. Examples of these two methods are provided.

NOTE: The Siebel Open UI client supports HTML markup only. For more information, see "Overview of the XML Web Interface" on page 41.

This topic contains the following information:

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 Account List View. The XML response from SWE is transformed into HTML using XSLT.

NOTE: For code snippets that demonstrate transforming an XML response from SWE into HTML, see "Sample XSLT" on page 104.

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

<%

'---------------------------------------------

'Open HTTP connection and send XML command req

'---------------------------------------------

   strURL = "http://" & Request.form ("swe") & "/
start.swe?SWECmd=ExecuteLogin&SWEDataOnly=1&SWEUserName=sadmin&SWEPassword=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=GotoView&SWEView=Account+List+View&SWESetMarkup=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";

   InputStream      in;
   BufferedReader   fromServer;
   PrintWriter      toServer;
   Socket           socket;
   String           payload;
   String           line;

   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());

}
```

XML Request Structure

The XML API offers developers access to the objects within Siebel Business 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.

NOTE: The Siebel Open UI client supports HTML markup only. For more information, see "Overview of the XML Web Interface" on page 41.

This topic contains the following information:

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 & character. 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 74.

XML Command Block

To initiate an action on a Siebel Business Applications XML screen, you must use a specific set of XML tags and they must conform to a specific structure. Table 19 lists the three valid XML tags that are used to perform a command.

Table 19. XML Tags

Tag	Description
<EXEC>	This is 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.
<CMD>	This tag indicates the SWE command that you want to access and encloses all of the 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 19, 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 topic describes the syntax of each XML tag. For the DTD for the inbound XML document, see "Inbound DTD" on page 90.

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. Use 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 20 lists the attribute used with the Execute tag:

Table 20. EXEC Tag Attribute

Attribute	Description
PATH	The PATH attribute is used to indicate the location of the SWE object manager. By default, the SWE XML application looks in its root directory for the SWE object manager. If you want to specify an Application Object Manager for the Web application to use, then 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. Use only one <CMD> tag for each command block that you want to execute.

Attributes

Table 21 lists the attributes that are used with the Command tag:

Table 21. CMD Tag Attributes

Attribute	Description
NAME	The NAME attribute must 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 Business: ■ ExecuteLogin ■ GotoPageTab ■ InvokeMethod ■ LogOff

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="SWESetNoTempl">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 does not execute without having all the appropriate parameters passed to the SWE.

Attributes

Table 22 lists the attributes that are used with the Argument tag.

Table 22. ARG Tag Attributes

Attribute	Description
NAME	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. Listed below are the parameter names most commonly used with Business: ■ SWEApplet ■ SWEDataOnly ■ SWEMethod ■ SWEPassword ■ SWEScreen ■ SWESetNoTempl ■ SWESetMarkup ■ SWESetRowCount ■ SWEStyleSheet ■ SWEUserName ■ SWEView Table 23 on page 53 lists the values that are most commonly used with these parameter names.

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, then you must send those parameters to the SWE. With the WriteRecord method, you must indicate the view and the applet that you are working with. You also must indicate the column to which you want to write the record, and you must 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="SWESetNoTempl">TRUE</ARG>

   </CMD>

</EXEC>
```

Required Arguments

The following 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>
```

For detailed information about these arguments, see the following:

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

  NOTE: SWESetMarkup is not used for the Siebel Open UI client, which supports HTML only.

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

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

- **SWESetNoTempl.** By default, Siebel Business Applications 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 returns 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, then the default is FALSE.

Common Name-Value Pairs

Table 23 lists commonly used argument name-value pairs.

Table 23. ARG Parameter Name-Value Pairs

Parameter Name	Parameter Values
SWEApplet	*Applet name*
SWEDataOnly	TRUE FALSE

Table 23. ARG Parameter Name-Value Pairs

Parameter Name	Parameter Values
SWEMethod	DeleteRecord EditRecord ExecuteQuery GoToNextSet GotoPageTab NewRecord NewQuery WriteRecord
SWEPassword	*Password*
SWEScreen	*Screen name*
SWESetMarkup	HTML XML
SWEUserName	*User name*
SWEView	*View name*

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

XML Response Structure

When you send a command block to a SWE XML application, you access the Siebel Business Applications XML application screens. If the action specified in the command block is successfully executed, then 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 topic, 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 90.

NOTE: The Siebel Open UI client supports HTML markup only. For more information, see "Overview of the XML Web Interface" on page 41.

This topic contains the following information:

XML Error Response

If a command block contains an error or is unsuccessful, then the specified action is not executed. Instead, the Siebel Business Applications XML user interface retains its current state and the SWE returns an error. Based on the markup format that 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 Business Applications 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 Business Applications XML application screen that you requested. The attributes within each tag are read-only and represent the properties of the object.

Table 24 on page 56 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 chapter are a subset of the tags that can be returned by the SWE.

Table 24. XML Response Tags

Tag	Description and Attributes
<APPLICATION>	The root tag for each response that is returned from the SWE. The <APPLICATION> tag encloses all the XML response data. Attribute: ■ **NAME.** This attribute indicates the name of the application from which the response is generated. For XML requests, the application name in the response is always *Siebel XML*.
<SCREEN>	This tag identifies the Siebel Business Applications screen that is the result of, or is accessed by, the command in your request. The <SCREEN> tag also encloses all of the XML tags that identify the data within the Siebel Business Applications screen. Attributes: ■ **CAPTION.** This attribute indicates the caption of the Siebel Business Applications screen. ■ **ACTIVE.** A value of TRUE indicates that the Siebel Business Applications screen is active. A value of FALSE indicates that the Siebel Business Applications screen is inactive. ■ **NAME.** This attribute indicates the Siebel Business Applications screen name, which is used to identify the Siebel Business Applications screen.
<VIEW>	This tag identifies the 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. Attributes: ■ **TITLE.** This attribute indicates the title of the view. ■ **ACTIVE.** A value of TRUE indicates that the view is active. A value of FALSE indicates that the view is inactive. ■ **NAME.** This attribute indicates the view name, which is used to identify the view.

Table 24. XML Response Tags

Tag	Description and Attributes
<APPLET>	This tag identifies the 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. Attributes: ■ **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, *1 - n of N*. ■ **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. ■ **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. ■ **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. ■ **MODE.** Indicates the mode of the applet, which can be one of the following: Base, Edit, New, Query, Sort. ■ **TITLE.** This attribute title of the applet. ■ **NO_INSERT.** A value of TRUE indicates that records cannot be inserted into the applet. ■ **CLASS.** Indicates the class being used by the applet. ■ **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. ■ **ACTIVE.** A value of TRUE indicates that the applet is active. A value of FALSE indicates that the applet is inactive. ■ **ID.** This attribute indicates the applet ID, and can be used to identify the applet. ■ **NAME.** This attribute indicates the applet name, which is used to identify the applet.
<LIST>	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 <LIST> tag: ■ <RS_HEADER> ■ <RS_DATA> There are no attributes associated with the <LIST> tag.

Table 24. 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>	A response can return multiple <COLUMN> tags. Each <COLUMN> tag within an <RS_HEADER> tag indicates another column within the parent list. Attributes: ■ **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. ■ **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. ■ **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. ■ **HTML_TYPE.** This attribute is used to indicate the type of object that is represented in the column. ■ **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. ■ **FIELD.** This attribute indicates the field name associated with the column. The value in the field name is the same as the column name. ■ **HIDDEN.** A value of TRUE indicates that the data in the column are hidden on the Siebel Business Applications screen. A value of FALSE indicates that the data are visible on the screen.

Table 24. XML Response Tags

Tag	Description and Attributes
<COLUMN>	■ **DATATYPE.** This attribute indicates the Siebel data-type of the data in the column. ■ **DISPLAY _NAME.** This attribute indicates the text string that would appear in the user interface. ■ **TEXT_LENGTH.** This attribute indicates the maximum length of field entries in the column. ■ **TYPE.** This attribute is used to indicate the type of object that is represented in the column. ■ **ID.** This attribute indicates the unique ID of the column. ■ **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. ■ **NAME.** A value of TRUE indicates that the data in the column are hidden on the Siebel Business Applications screen. A value of FALSE indicates that the data are visible on the screen. ■ **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. ■ **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.
<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 24. XML Response Tags

Tag	Description and Attributes
<ROW>	A response can return multiple <ROW> tags. Each <ROW> tag within an <RS_DATA> tag indicates another record within the table. The <ROW> tag encloses the <FIELD> tag. Attributes: ■ **SELECTED.** 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. ■ **ROWID.** This attribute is used to identify the row.
<FIELD>	A response can return multiple <FIELD> tags. Each <FIELD> tag within a <ROW> tag indicates another item of data within the record. The field's value is entered between the open and close <FIELD> tags. Attributes: ■ **VARIABLE.** This attribute indicates the column to which the field is associated. The value of the VARIABLE attribute coincides with the NAME attribute of a column. ■ **NAME.** This attribute is used to identify the field. In most cases, the field name is identical to the column name.

XML Response Syntax

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 Business Applications 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 Business Applications XML Web server is going to be in WML format.

Common Operations

You can use various combinations of XML commands to execute an action in a Siebel Business Applications XML application. Each of the following topics offers one solution for executing a Siebel Business Applications action:

- "Logging In" on page 62
- "Logging Off" on page 63
- "Navigating to a Screen" on page 63
- "Navigating Within a Screen" on page 64
- "Querying Items" on page 65
- "Modifying Records" on page 67
- "Deleting Records" on page 70
- "Picking Records" on page 71

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://<computer name>/callcenter/start.swe`. `<computer name>`. This is the Web server where the Siebel Business Applications are installed.

NOTE: The Siebel Open UI client supports HTML markup only. For more information, see "Overview of the XML Web Interface" on page 41.

Logging In

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

The following 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 must be a Logoff command.

The following 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 24 on page 56.

```
<?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 InvokeMethod to execute an XML command, you must also indicate the view and the applet that you want to access. For example, you might 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 24 on page 56. The example below shows 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 SWE XML 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 that 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 identify 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 SWE XML 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 new record.

```
<?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 SWE XML application:

1. Activate a new query. See "NewQuery" on page 65.
2. Execute the query. See "ExecuteQuery" on page 68.
3. Activate the edit record method. See "EditRecord" on page 68.
4. Write the record. See "WriteRecord" on page 69.

NOTE: When modifying a record, 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, then several records might be returned in the response. There is a chance that the record that 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 that 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 that 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 screen is populated with the record that 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, then several records might 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 SWE XML application:

1. Activate a new query. See "NewQuery" on page 70.
2. Execute the query. See "ExecuteQuery" on page 70.
3. Delete the selected record. See "DeleteRecord" on page 71.

NOTE: When deleting a record, 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, then several records might be returned in the response. There is a chance that the record that 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. 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 that 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 that 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 save the value in the database, you need to navigate to a screen and then submit three requests:

1. Navigate to a screen. See "GotoPageTab" on page 72.
2. Get a pick list. See "EditField" on page 72.
3. Get the row ID of the record to pick. See "PickRecord" on page 73.

4 Write the record to the database. See "WriteRecord" on page 73.

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 row ID 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 the 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 topic contains reference information about SWE commands, methods, and arguments:

NOTE: The Siebel Open UI client supports HTML markup only. For more information, see "Overview of the XML Web Interface" on page 41.

SWE Commands

Table 25 on page 75 provides a list of commonly used SWE commands.

NOTE: A subset of these SWE commands are available in Siebel Open UI. These commands are listed in Table 26 on page 79.

Table 25. SWE Commands

Supported Values	Short Format	Description	Required Args (with Description)	Optional Args (with Description)
CanInvokeMethod For a list of commonly used methods, see Table 27 on page 80.	C	Checks whether a method can be invoked on an applet, a business service, a business component, or the SWE application. Called only when the Object Manager is in high interactivity mode. Use the optional **SWEService**, **SWEBusComp**, and **SWEApplet** arguments to specify the Siebel object on which the method is invoked. If none of these are specified, then SWE checks 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.	**SWEMethod** - name of the method.	**SWEService** - nameof the business service to check whether the method can be invoked. **SWEBusComp** - name of the business component to check whether the method can be invoked. **SWEApplet** - name of the applet to check whether the method can be invoked.
ExecuteLogin	Xlg	Executes login for a user.	**SWEUserName** - user name. **SWEPassword** - password.	None
GotoPage	Gp	Goes to a Siebel Web page (this is the Web page object defined in Siebel Tools).	**SWEPage** - name of the Web page.	None
GotoPageTab	Gt	Goes to a Siebel screen. Shows the default view for the screen.	**SWEScreen** - name of the screen.	None

Table 25. SWE Commands

Supported Values	Short Format	Description	Required Args (with Description)	Optional Args (with Description)
GotoView	Gv	Goes to a Siebel view. If the **SWEPostnApplet** and **SWEPostnRowId** arguments are specified, then it executes a search for the specified row ID in the specified applet. **NOTE:** If the queried applet is part of a toggle cycle, then set SWEPostnApplet to the default (top) applet in the toggle cycle or the application displays an error, View: %1 does not contain applet: %2. For more information about applet toggles, see *Configuring Siebel Business Applications*. If **SWEQMApplet** and **SWEQMMethod** arguments are specified, then it invokes the method after going to the view.	**SWEView** - name of the view.	**SWEKeepContext** - if TRUE, keeps the current business object context, when requesting to a view based on the same business object. **SWEPostnApplet** - name of the applet on which to execute the search. **SWEPostnRowId** - row ID to search for. **SWEQMApplet** - name of the QueueMethod applet where the method (as specified in **SWEQMMethod**) is invoked after going to the view. **SWEQMMethod** - name of the QueueMethod method to be invoked. You can invoke only one method. **SWEQMArgs** - arguments of the QueueMethod method.

Table 25. SWE Commands

Supported Values	Short Format	Description	Required Args (with Description)	Optional Args (with Description)
InvokeMethod For a list of commonly used methods, see Table 27 on page 80.	Inv	Invokes a method on an applet, a business service, a business component, or the SWE application. Use the optional **SWEService**, **SWEBusComp**, and **SWEApplet** arguments to specify the Siebel object on which the method is invoked. If none of these are specified, then SWE invokes on the SWE application object, which currently supports a limited set of InvokeMethod such as Logoff, SortOrder, SaveQuery, and SaveQueryAs.	**SWEMethod** - name of the method.	**SWEService** - nameof the business service to invoke the method. **SWEBusComp** - name of the business component to invoke the method. **SWEApplet** - name of the applet to invoke the method. **SWEView** - name of the view to invoke the method
LoadService	None	Loads a business service on the server side.	**SWEService** - 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, then the default is the "Acknowledgment Web Page" property to show the login page.	None	None

Table 25. 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, then SWE shows the login view or login page, depending on how you log in.	None	None
ReloadCT	None	Reloads personalization info. SWE loads the initial personalization on startup, and when the personalization rulesare 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

SWE Commands Available in Siebel Open UI

You can use several SWE commands to display a Siebel portlet in Siebel Open UI. For security reasons, you can use only the GotoViewmethod to call a Siebel portlet from an external application. GotoPage and GotoPageTab are not applicable to Siebel Open UI. You can use the commands listed in Table 26 within a Siebel portlet. You cannot use them to call a portlet. For more information about these commands, see *Configuring Siebel Open UI*.

Table 26. SWE Commands Available in Siebel Open UI

Supported Values	Inside External Application	Called from UI Element Inside Portlet Container	Called from Outside Portlet Container
CanInvokeMethod	Yes	Yes	No
ExecuteLogin	Yes	Not applicable for this use case.	No
GotoView	Use only when invoked from the browser address bar by refresh or history navigation.	Yes	Yes
InvokeMethod	Yes	Yes	No
LoadService	Yes	Yes	No
Login	Yes	Not applicable to Siebel Open UI.	Not applicable (use SSO or similar)
Logoff	Yes	Not applicable to Siebel Open UI.	No
ReloadCT	Yes	Yes	No

SWE Methods

The InvokeMethod command allows you to invoke SWE methods on an applet, business component, business service, or application. Table 27 lists SWE methods commonly used with the InvokeMethod SWE command.

Table 27. 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.	**SWETreeItem**: Specify the path of the item relative to root. The path is a string of the form *n*.*n*.*n*.*n*...where *n* is an index of an item within its level. The index starts from 1. Example: 1.1.2. **SWEView**: Name of the view. **SWEApplet**: 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.	**SweNamedQueries**: 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 **SWEField**.	**SWEField**: Specify the name of the applet field that you want to drilldown on. The drilldown information is specified in the repository.	None

Table 27. SWE Methods

Supported Values	Description	Required Args (with Description)	Optional Args (with Description)
EditRecord	Changes the Applet Web Template from base mode to edit mode, so the record can be edited. Use EditRecord with applets running in standard interactivity. For applets running in high interactivity, it is not necessary tochange the Applet Web Template mode to edit the record. For high interactivity applets, use WriteRecord. Siebel Open UI uses high interactivity applets.	**SWESeq**: 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. Sets the 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	None
ExecuteNamedQuery	Executes a predefined query (PDQ) on the current view. Use with standard interactivity applications.	**SWEQueryName** - name of the PDQ.	None
ExpandTreeItem	Used in a tree control to expand an item on the tree.	**SWETreeItem**: Specify the path of the item relative to root. The path is a string of the form *n*.*n*.*n*.*n*...where *n* is an index of an item within its level. The index starts from 1. Example: 1.1.2. **SWEView**: Name of the view. **SWEApplet**: 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

Table 27. SWE Methods

Supported Values	Description	Required Args (with Description)	Optional Args (with Description)
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
GotoView	Goes to a Siebel view. If the **SWEPostnApplet** and **SWEPostnRowId** arguments are specified, then this method executes a search for the specified row ID in the specified applet. **NOTE:** If the queried applet is part of a toggle cycle, then set SWEPostnApplet to the default (top) applet in the toggle cycle or the application displays an error, View: %1 does not contain applet: %2. For more information about applet toggles, see *Configuring Siebel Business Applications*. If **SWEQMApplet** and **SWEQMMethod** arguments are specified, then this method invokes the method after going to the view.	**SWETargetView** - name of the view.	**SWEKeepContext** - if TRUE, then this method keeps the current business object when the user navigates to a view that uses the same business object. **SWEPostnApplet** - name of the applet on which to execute the search. **SWEPostnRowId** - row ID to search for. **SWEQMApplet** - name of the QueueMethod applet where the method (as specified in **SWEQMMethod**) is invoked after going to the view. **SWEQMMethod** - name of the QueueMethod method. The method to be invoked. You can invoke only one method. **SWEQMArgs** - arguments of the QueueMethod method.

Table 27. SWE Methods

Supported Values	Description	Required Args (with Description)	Optional Args (with Description)
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
NewQuery	Begins a new query.	None	None
NewRecord	If the applet has an association applet, then this method shows the association popup applet. Otherwise, it creates a new record.	None	None
NextTreeItem	Used in a tree control to scroll the tree to the next set of record.	**SWETreeItem**: Specifies the path of the item relative to root. The path is a string of the form *n*.*n*.*n*.*n*...where *n* is an index of an item within its level.The index starts from 1. Example: 1.1.2. SWEView: Name of the view. **SWEApplet**: 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

Table 27. SWE Methods

Supported Values	Description	Required Args (with Description)	Optional Args (with Description)
PickRecord	Picks the current row in a pick applet.	None	None
PositionOnRow	Positions the record as specified in the list of required arguments.	**SWEView**: Name of the view. **SWEApplet**: Name of the Applet. **SWERowId**: The row ID of the desired record. **SWESetRowCnt**: Sets the number of rows to be returned for XML requests. When used during PositionOnRow, the specified number of rows are returned, and the selected row remains highlighted. **SWEReqRowId**: Indicates that the row is required in the operation.	None
PostChanges	Sets the field values as specified in the list of arguments to therecord 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. Sets these field values before committing the record.
PreviousTreeItem	Used in a tree control to scroll the tree to the previous set of records.	**SWETreeItem**: Specify the path of the item relative to root. The path is a string of the form *n*.*n*.*n*.*n*...where *n* is an index of an item within its level. The index starts from 1. Example: 1.1.2. SWEView: Name of the view. **SWEApplet**: Name of the applet.	None

Table 27. SWE Methods

Supported Values	Description	Required Args (with Description)	Optional Args (with Description)
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 **_SweNamedQueries**.	**SweNamedQueries**: Specify the name tosave the query as.	None
SelectTreeItem	Used in a tree control to select an item of the tree.	**SWETreeItem**: Specifies the path of the item relative to root. The path is a string of the form *n*.*n*.*n*.*n*...where *n* is an index of an item within its level.The index starts from 1. Example: 1.1.2. SWEView: Name of the view. **SWEApplet**: Name of the applet.	None
SortAscending	Sorts the field as specified in the argument **SWEField** in ascending order.	**SWEField**: 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 **SWEField** in descending order.	**SWEField**: Specifies the name of the applet field that you want to sort in descending order.	None
ToggleTo	Toggles to a different toggle applet.	**SWESeq**: Sequence number of the toggle 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.	**SWERowId**: Is the row ID of the record to be saved. **SWEReqRowId**: Indicates that the row ID is required in the operation.	None

SWE Arguments

Table 28 lists some commonly used SWE arguments.

Table 28. SWE Arguments

URL Argument	Short Format	Description	Usage	Examples
SWEAC	None	Allows login manager to string two SWE commands in a single request. (Formerly known as SWEAuxCmd.)	SWECmd=ExecuteLogin (followed by) SWEAC=GotoPageTab	SWECmd=ExecuteLogin&SWEUserName=joe&SWEPassword=passwd&SWEAC=SWECmd=GotoPageTab&SWEScreen=Accounts+Screen&SWEReloadFrames=1
SWEBU	None	Indicates that a URL is a bookmarked URL. It is retrieved in the UI by using the Get Bookmark URL command.	SWEBU=1 (if used as a bookmark URL)	None
SWECount	C	Dynamically generates an index number for each hyperlink for the purpose of bookmarking each request.	SWEC=*n* (where *n* is a positive integernumber) (or) <ARG NAME="SWEC">*n*</ARG>	SWEC=1 (or) <ARG NAME="SWEC">1</ARG>
SWEDataOnly	None	Discards all UI content (including anchors) if set to TRUE.	SWEDataOnly={TRUE|FALSE} (or) <ARG NAME=”SWEDataOnly”>TRUE|FALSE</ARG>	SWEDataOnly=TRUE (or) <ARG NAME=”SWEDataOnly”>TRUE</ARG>
SWEExclude	None	Uses the comma-separated UI element names specified as the value of the parameter to exclude UI elements from appearing in the output document.	SWEExclude=”list of names” (names can be MENU, SCREENBAR, TOOLBAR, THREADBAR, PAGEITEM, VIEWBAR) (or) <ARG NAME=”SWEExclude”>list of names</ARG>	SWEExclude=”MENU,SCREENBAR” (or) <ARG NAME=”SWEExclude”>MENU,SCREENBAR</ARG>

Table 28. SWE Arguments

URL Argument	Short Format	Description	Usage	Examples
SWEField	F	Specifies the name of the applet field.	`SWEField=<field name>` (or) `<ARG NAME="SWEField">field name</ARG>`	`SWEField=Revenue` (or) `<ARG NAME="SWEField">Revenue</ARG>`
SWEFullRefresh	None	Forces a full refresh of the Siebel Web Client, for applications deployed in Siebel Open UI or high interactivity. Used by the Siebel Open UI or high interactivity client to send a SWE command to load the completely. Typically used for session interleaving from a non-Siebel session.	`SWEFullRefresh={TRUE|FALSE}` (or) `<ARG NAME="SWEFullRefresh">TRUE|FALSE</ARG>`	`SWEFullRefresh=TRUE` (or) `<ARG NAME="SWEFullRefresh">TRUE</ARG>`
SWEGetApplet	None	This parameter is used to filter the outbound XML document so only the applet named as the value of the parameter is allowed in the output. All other document content is discarded.	`SWEGetApplet=<name of the applet>` (or) `<ARG NAME="SWEGetApplet">name of the applet</ARG>`	`SWEGetApplet=Account+List+Applet` (or) `<ARG NAME="SWEGetApplet">Account List Applet</ARG>`
SWEGetPDQ	None	Discards all XML content and returns only PDQ list when set to TRUE.	`SWEGetPDQ={TRUE|FALSE}` (or) `<ARG NAME="SWEGetPDQ">TRUE|FALSE</ARG>`	`SWEGetPDQ=TRUE` (or) `<ARG NAME="SWEGetPDQ">TRUE</ARG>`

Table 28. SWE Arguments

URL Argument	Short Format	Description	Usage	Examples
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}` (or) `<ARG NAME=”SWEKeepContext”>TRUE|FALSE</ARG>`	`SWEKeepContext=TRUE` (or) `<ARG NAME=”SWEKeepContext”>TRUE</ARG>`
SWENeedContext	Nct	Skips restoring the state of the view, applet, business object, and business component 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}` (or) `<ARG NAME=”SWENeedContext”>TRUE|FALSE</ARG>`	`SWENeedContext=TRUE` (or) `<ARG NAME=”SWENeedContext”>TRUE</ARG>`
SWENoAnchor	None	Discards all anchors if set to TRUE.	`SWENOAnchor={TRUE|FALSE}` (or) `<ARG NAME=”SWENoAnchor”>TRUE|FALSE</ARG>`	`SWENoAnchor=TRUE` (or) `<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 SWERowId, if set to TRUE.	`SWEReqRowId={TRUE|FALSE}` (or) `<ARG NAME=”SWEReqRowId”>TRUE|FALSE</ARG>`	`SWEReqRowId=TRUE` (or) `<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 integernumber) (or) `<ARG NAME="SWERs">n</ARG>`	`SWERs=1` (or) `<ARG NAME="SWERs">1</ARG>`

Table 28. SWE Arguments

URL Argument	Short Format	Description	Usage	Examples
SWERowId	R	The row ID of the record to position to.	SWERowId=<rowid><ARG NAME="SWERowId"> rowid</ARG>	SWERowId=12-XI46FG<ARG NAME="SWERowId">12-XI46FG</ARG>
SWERowIds	Rs	A string specifying the row ID of the parent business components.	SWERowIds=<string of rowids><ARG NAME="SWERowId"> string of rowids</ ARG>	SWERowIds=SWERowId0%3d12-61W25L<ARG NAME="SWERowId">SWERowId=12-61W25L</ ARG>
SWESetMarkup	None	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	None	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	None	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	None	Specifies the name of the XSLT style sheet to use to perform the XSLT on the XML output document.	SWEXslStyleSheet=<stylesheet name> (The style sheet 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 topic lists Document Type Definitions (DTD) for the inbound and outbound documents used with the XML Web Interface.

NOTE: The Siebel Open UI client supports HTML markup only. For more information, see "Overview of the XML Web Interface" on page 41.

Inbound DTD

The following is the DTD for the inbound documents used with the XML Web Interface.

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

   ATTR     CDATA   #IMPLIED
   PATH     CDATA   #IMPLIED
   TARGET   CDATA    #IMPLIED

>
<!ELEMENT   CMD                  (ARG*) >
<!ATTLIST   CMD

   NAME     CDATA   #REQUIRED
   VALUE    CDATA   #REQUIRED

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

   NAME     CDATA   #REQUIRED

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

   NAME     CDATA   #REQUIRED

>
```

Outbound DTD

The following is the DTD for the outbound documents used with the XML Web Interface.

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

      NAME  CDATA                 #REQUIRED

>
<!ELEMENT   USER_AGENT           EMPTY>
<!ATTLIST   USER_AGENT

   MARKUP   CDATA                #REQUIRED
   TYPE     CDATA                 #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

   NAME    CDATA                #REQUIRED
   VALUE    CDATA                #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

   NAME     CDATA                 #REQUIRED
   PATH     CDATA                  #IMPLIED

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

   NAME       CDATA                #REQUIRED
   TYPE       CDATA                #REQUIRED
   ATTR       CDATA                #IMPLIED
   MAX_LENGTH CDATA#IMPLIED
```

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

   NAME    CDATA                  #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

   MODE    CDATA                  #IMPLIED
   SCREEN  CDATA                  #IMPLIED
   TYPE    CDATA                   #IMPLIED

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

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

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

   NAME    CDATA                  #REQUIRED
   ATTR    CDATA                  #IMPLIED
   CAPTION CDATA                  #IMPLIED
   TYPE    CDATA                   #REQUIRED

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

   NAME    CDATA                  #REQUIRED
   ACTIVE  (TRUE | FALSE)         "FALSE"
   CAPTION  CDATA                  #IMPLIED

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

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

```
>
<!ELEMENT   APPLET          (FORM | CONTROL | CALENDAR | TREE | (LIST | (RS_HEADER,
                            RS_DATA)) | SORT_FIELD | APPLET_TOGGLE | ERROR)*  >

<!ATTLIST   APPLET

    NAME           CDATA            #REQUIRED
    ACTIVE         CDATA            #IMPLIED
    CLASS          CDATA            #IMPLIED
    ID             CDATA            #IMPLIED
    MODE           CDATA            #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_COUNTER    CDATA            #IMPLIED
    TITLE           CDATA            #IMPLIED

>
<!ELEMENT   FORM         ((CONTROL | CALENDAR | TREE | (LIST | (RS_HEADER, RS_DATA))
                 | SORT_FIELD | APPLET_TOGGLE | 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

    NAME            CDATA          #REQUIRED
    CALCULATED      (TRUE | FALSE)  "FALSE"
    DISPLAY_NAME    CDATA          #IMPLIED
    DATATYPE        CDATA          #IMPLIED
    FIELD           CDATA          #IMPLIED
    FORMAT          CDATA          #IMPLIED
    HIDDEN          (TRUE | FALSE)  "FALSE"
    HTML_TYPE       CDATA          #IMPLIED
    ID              CDATA          #IMPLIED
    LIST_EDITABLE   CDATA          #IMPLIED
    NUMBER_BASED    (TRUE | FALSE)  "FALSE"
    READ_ONLY       (TRUE | FALSE)  "FALSE"
    REQUIRED        (TRUE | FALSE)  "FALSE"
    SCALE           CDATA          #IMPLIED
    TEXT_ALIGN      CDATA          #IMPLIED
    TEXT_BASED      (TRUE | FALSE)  "FALSE"
```

```
    TEXT_LENGTH     CDATA           #IMPLIED
    TOTAL_REQUIRED  (TRUE | FALSE)  "FALSE"
    TYPE             CDATA           #IMPLIED

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

    ROWID    CDATA                   #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   APPLET_TOGGLE          (TOGGLE_ITEM | ERROR)* >
<!ATTLIST   APPLET_TOGGLE

   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 Style Sheets 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 style sheets used for the transformation and specify the names of the style sheets to SWE.

NOTE: The Siebel Open UI client supports HTML markup only. For more information, see "Overview of the XML Web Interface" on page 41.

This topic contains the following information:

Defining SWE Style Sheet 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 style sheets to use in the Siebel templates by means of the `<swe:xsl-stylesheet>` tag. For more information, see "XML-Specific Template Tag" on page 97.

XML-Specific Template Tag

The XML-specific template tag looks like this:

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

Purpose

Specifies the name of the XSLT style sheet to perform the XSLT on the XML output document. The style sheet 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, then the last tag found gets used.

Attributes

Two attributes are used with the XSLT style sheet:

- **name.** Specifies the name of the style sheet.
- **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 this attribute is set to *embed*, SWE inserts an XML processing instruction in the beginning of the XML document for external XSLT processing.

Example

The following example illustrates how to specify the attributes for a style sheet.

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

Sample XSL Style Sheet

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

Sample XSLT

The following example shows how XSLT code snippets transform an XML response from SWE into HTML. The XSLT snippets are based on the XML response generated from the Query String example described in "Connecting to the XML Web Interface" on page 44.

```
<xsl:template match="/">
   <TABLE bgcolor="#CCCCFF" width="100%" cellpadding="2"
   cellspacing="0" Border="0" >
   <TBODY>
      <xsl:apply-templates select="//APPLET/LIST"/>
   </TBODY>
   </TABLE>
</xsl:template>

<xsl:template match="LIST">
   <xsl:apply-templates select="RS_HEADER"/>
   <xsl:apply-templates select="RS_DATA"/>
</xsl:template>

<xsl:template match="RS_HEADER">
   <TR>
      <xsl:for-each select="COLUMN">
      <xsl:if test="@NAME='Name'">
         <TD colspan="3" bgcolor="#CCCCFF" class="sub2viewon" width="60%">
         <B><xsl:value-of select="@DISPLAY_NAME"/></B
         </TD>
      </xsl:if>
      <xsl:if test="@NAME='Location'">
         <TD bgcolor="#CCCCFF" class="sub2viewon" width="40%">
         <B><xsl:value-of select="@DISPLAY_NAME"/></B>
         </TD>
      </xsl:if>
      </xsl:for-each>
   </TR>
</xsl:template>

<xsl:template match="RS_DATA">
   <xsl:for-each select="ROW">
      <TR>
         <xsl:for-each select="FIELD">
         <xsl:if test="@NAME='Name'">
            <TD bgcolor="#FFFFFF">
               <xsl:element name="IMG">
                  <xsl:attribute name="SRC">
                     portal_files/w.gif
                  </xsl:attribute>
```

```
            <xsl:attribute name="height">
               1
            </xsl:attribute>
            <xsl:attribute name="width">
               3
            </xsl:attribute>
         </xsl:element>
      </TD>
      <TD bgcolor="#FFFFFF" valign="top">
         <xsl:element name="IMG>
            <xsl:attribute name="SRC">
               portal_files/dot.gif
            </xsl:attribute>
            <xsl:attribute name="height">
               6
            </xsl:attribute>
            <xsl:attribute name="width">
               6
            </xsl:attribute>
         </xsl:element>
      </TD>
      <TD bgcolor="#FFFFFF" align="left" valign="top"
      width="60%">
         <xsl:choose>
            <xsl:when test="string-length(normalize
            space(.))> 0"
               <xsl:choose>
                  <xsl:when test="@NAME='Name'">
                  <xsl:call-template name="link"/>
                  </xsl:when>
                  <xsl:otherwise>
                  <xsl:value-of select="."/>
                  </xsl:otherwise>
               </xsl:choose>
            </xsl:when>
            <xsl:otherwise>
            <xsl:text> </xsl:text>
            </xsl:otherwise>
         </xsl:choose>
      </TD>
   </xsl:if>
   <xsl:if test="@NAME='Location'">
      <TD bgcolor="#FFFFFF" align="left" valign="top"
      width="40%">
         <xsl:choose>
            <xsl:when test="string-length(normalize-space(.))
            < 0">
               <xsl:choose>
                  <xsl:when test="@NAME='Name'">
                  <xsl:call-template name="link"/>
                  </xsl:when>
                  <xsl:otherwise>
                  <xsl:value-of select="."/>
                  </xsl:otherwise>
```

```
                  </xsl:choose>
                </xsl:when>
                <xsl:otherwise>
                <xsl:text> </xsl:text>
                </xsl:otherwise>
              </xsl:choose>
            </TD>
          </xsl:if>
          </xsl:for-each>
        </TR>
        <tr>
          <td colspan="4" width="40%"></td>
        </tr>
    </xsl:for-each>
</xsl:template>
```

5 Web Engine HTTP TXN Business Service

This chapter describes the Web Engine HTTP TXN Business Service. It contains the following information:

About the Web Engine HTTP TXN Business Service

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 allows Siebel Business 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 business service 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 business service can also set a custom cookie or header in the HTTP response header generated by the SWE. The business service gives complete control over the request header received and the response header sent by the SWE.

For more information, see the following topics:

- "Web Engine HTTP TXN Business Service API" on page 108
- "Example of Using Web Engine HTTP TXN Business Service" on page 111
- "Sample Code for Invoking Methods of Web Engine HTTP TXN Business Service" on page 116

Web Engine HTTP TXN Business Service API

Table 29 lists the methods exposed by the Web Engine HTTP TXN Business Service.

Table 29. Web Engine HTTP TXN Business Service API

Method	Description	Parameters
GetAllRequestCookies	Retrieves all request cookies sent from the client to the server.	**InputArguments:** Ignored. **OutputArguments:** 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.	**InputArguments:** Ignored. **OutputArguments:** Property Set containing the HTTP Parameter name-value pairs.
GetAllRequestParameters	Retrieves all request parameters sent from the client to the server.	**InputArguments:** Ignored. **OutputArguments:** Property Set containing the HTTP Parameter name-value pairs.
GetAllResponseCookies	Retrieves all response cookies sent from the server to the client.	**InputArguments:** Ignored. **OutputArguments:** 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.	**InputArguments:** Ignored. **OutputArguments:** Property Set containing the HTTP Header name-value pairs.
GetAllServerVariables	Retrieves all server variables.	**InputArguments:** Ignored. **OutputArguments:** Property Set containing the Server Variable name-value pairs.
GetClientCertificate	Retrieves the client certificate info.	**InputArguments:** Ignored. **OutputArguments:** Property Set containing certificate name-value pairs. Currently only returns Common Name (CN) property of the certificate.

Table 29. Web Engine HTTP TXN Business Service API

Method	Description	Parameters
GetRequestCookies	Retrieves the request cookies named in InputArguments.	**InputArguments:** Property Set containing the cookie names to retrieve. **OutputArguments:** 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.	**InputArguments:** Property Set containing the header names to retrieve. **OutputArguments:** 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.	**InputArguments:** Ignored **OutputArguments:** 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.	**InputArguments:** Property Set containing the parameter names to retrieve. **OutputArguments:** Property Set containing the HTTP Parameter name-value pairs.
GetResponseCookies	Retrieves the response cookies named in InputArguments.	**InputArguments:** Property Set containing the cookie names to retrieve. **OutputArguments:** Property Set hierarchy. Each cookie is a child Property Set with the TYPE property set to the cookie name.
GetResponseHeaders	Retrieves the response headers named in InputArguments.	**InputArguments:** Property Set containing the header names to retrieve. **OutputArguments:** Property Set containing the HTTP Header name-value pairs.

Table 29. Web Engine HTTP TXN Business Service API

Method	Description	Parameters
GetResponseInfo	Retrieves the response Headers and Cookies in one call.	**InputArguments:** Ignored. **OutputArguments:** Property Set hierarchy. Each section is a child Property Set with the TYPE property set to 'Headers' or 'Cookies'. Content Type and Status aresimply stored as properties of OutputArguments.
GetServerVariables	Retrieves the server variables named in InputArguments.	**InputArguments:** Property Set containing the server variable names to retrieve. **OutputArguments:** Property Set containing the Server Variable name-value pairs.
GetWebSessionInfo	Retrieves the client's Web session information.	**InputArguments:** Ignored. **OutputArguments:** Property Set containing the Web session name-value pairs—SessionName; Cookie Name; SessionId; Web Session ID; SessionFrom (Value is 'URL' or 'COOKIE').
SetResponseCookies	Sets the response cookies to the values in InputArguments.	**InputArguments:** 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 cookiepersists between sessions. If the value is Y, then the cookie persists between browser sessions. Otherwise, the cookie exists for one session at a time. **OutputArguments:** Ignored.

Table 29. Web Engine HTTP TXN Business Service API

Method	Description	Parameters
SetResponseHeaders	Sets the response headers to the values in InputArguments.	**InputArguments:** Property Set containing the HTTP Header name-value pairs. **OutputArguments:** Ignored.
SetResponseInfo	Sets the response Headers and Cookies in one call.	**InputArguments:** 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. **OutputArguments:** Ignored.

Example of Using Web Engine HTTP TXN Business Service

To invoke each method of the Web Engine HTTP TXN Business Service and write the results to a text file, use the following two procedures:

- "Adding Sample Code for Displaying Results of Using the Business Service" on page 111
- "Adding Sample Code for Invoking Methods of the Business Service" on page 113

Adding Sample Code for Displaying Results of Using the Business Service

The following procedure shows how to add sample code for displaying results of the Web Engine HTTP TXN Business Service.

To add sample code for displaying results of Web Engine HTTP TXN Business Service

1. In Oracle's Siebel Tools, navigate to the desired Applet object, in the Object Explorer.
2. Lock the project, if required.
3. Right click and select the Edit Server Script option.
4. Add the following three functions, individually to the declarations section:
   - WebApplet_OutputChildPropertySets
   - WebApplet_OutputProperties
   - WebApplet_OutputPropertySet

Sample Code Functions

Sample code for the WebApplet_OutputChildPropertySets Function:

```
function WebApplet_OutputChildPropertySets(oPropertySet, nLevel, fp)
{
var oChildPropSet;
var nChild = 0;

Clib.fputs('-------------------------------------\n',fp);
Clib.fputs('CHILD PROPERTY SETS\n',fp);
Clib.fputs('-------------------------------------\n',fp);

if ( oPropertySet.GetChildCount() == 0 )
{

Clib.fputs('(NONE)\n',fp);
}
else
{
for ( nChild = 0; ( nChild <= oPropertySet.GetChildCount() - 1 ) ; nChild++ )
{
oChildPropSet = oPropertySet.GetChild(nChild);
WebApplet_OutputPropertySet (oChildPropSet, nLevel+1, fp);
}
}
}
```

Sample code for the WebApplet_OutputProperties Function:

```
function WebApplet_OutputProperties(oPropertySet, nLevel , fp )
{
var strName;
var strValue;

Clib.fputs('-------------------------------------\n',fp);
Clib.fputs('PROPERTIES\n',fp);
Clib.fputs('-------------------------------------\n',fp);

if (oPropertySet.GetPropertyCount() == 0 )
{
Clib.fputs('(NONE)\n',fp);
}
else
{
strName = oPropertySet.GetFirstProperty();
while ( strName != '')
{
Clib.fputs(strName + ' : ' + oPropertySet.GetProperty(strName) + '\n' ,fp);
strName = oPropertySet.GetNextProperty();
}
}
}
```

Sample code for the WebApplet_OutputPropertySet Function:

```
function WebApplet_OutputPropertySet(oPropertySet, nLevel, fp )
{
Clib.fputs('\n',fp);
Clib.fputs('-------------------------------------\n',fp);
Clib.fputs('START' + ' ',fp);
Clib.fputs('LEVEL : ' + nLevel + '\n', fp);
Clib.fputs('-------------------------------------\n',fp);

Clib.fputs('TYPE  : ' + oPropertySet.GetType() + '\n',fp);
Clib.fputs('VALUE : ' + oPropertySet.GetValue() + '\n',fp);

WebApplet_OutputProperties(oPropertySet, nLevel, fp);
WebApplet_OutputChildPropertySets(oPropertySet, nLevel, fp);

Clib.fputs('-------------------------------------\n',fp);
Clib.fputs('END' + ' ',fp);
Clib.fputs('LEVEL : ' + nLevel + '\n',fp);
Clib.fputs('-------------------------------------\n',fp);
}
```

Adding Sample Code for Invoking Methods of the Business Service

The following procedure shows how to add sample code for invoking methods of the Web Engine HTTP TXN Business Service.

To add sample code for invoking methods of Web Engine HTTP TXN Business Service

1. Add the code from "Sample Code for Invoking Methods of Web Engine HTTP TXN Business Service" on page 116 to the WebApplet_InvokeMethod event.
2. Compile the project.
3. Start the Siebel application.
4. Navigate to the applet where the server script has been placed.
5. Perform an action on the applet that invokes a SWE method (for example, change the record or create a new record).

   The code generates a text file in the bin directory where the Siebel application is installed containing results of each method of the Web Engine HTTP TXN Business Service.

Sample Output

The following is an excerpt of the resulting text file.

```
====================================
WebApplet InvokeMethod event:
====================================

====================================
Method: GetAllRequestCookies
====================================
```

```
-------------------------------------
START LEVEL : 0
-------------------------------------
TYPE  : COOKIES
VALUE :
-------------------------------------
PROPERTIES
-------------------------------------
(NONE)
-------------------------------------
CHILD PROPERTY SETS
-------------------------------------

-------------------------------------
START LEVEL : 1
-------------------------------------
TYPE  : SWEUAID
VALUE : 1
-------------------------------------
PROPERTIES
-------------------------------------
Max-Age : -1
Domain :
Path :
-------------------------------------
CHILD PROPERTY SETS
-------------------------------------
(NONE)
-------------------------------------
END LEVEL : 1
-------------------------------------
-------------------------------------
END LEVEL : 0
-------------------------------------

=====================================
Method: GetAllRequestHeaders
=====================================

-------------------------------------
START LEVEL : 0
-------------------------------------
TYPE  : HEADERS
VALUE :
-------------------------------------
PROPERTIES
-------------------------------------
HOST : <host computer name>
CACHE-CONTROL : no-cache
CONNECTION : Keep-Alive
COOKIE : SWEUAID=1
USER-AGENT : Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; Q312461; SV1; .NET
CLR 1.1.4322)
CONTENT-TYPE : application/x-www-form-urlencoded
```

```
ACCEPT-ENCODING : deflate
CONTENT-LENGTH : 348
-------------------------------------
CHILD PROPERTY SETS
-------------------------------------
(NONE)
-------------------------------------
END LEVEL : 0
-------------------------------------

=====================================
Method: GetAllRequestParameters
=====================================

-------------------------------------
START LEVEL : 0
-------------------------------------
TYPE  : PARAMETERS
VALUE :
-------------------------------------
PROPERTIES
-------------------------------------
SWEActiveView : Account List View
SWERowIds :
SWEP :
SWESP : false
SWECmd : InvokeMethod
SWEMethod : PositionOnRow
SWER : 1
SWEControlClicked : 0
SWEIgnoreCtrlShift : 0
SWEVI :
SWEActiveApplet : Account List Applet
SWERPC : 1
SWEReqRowId : 1
SWEView : Account List View
SWEC : 3
SWERowId : 1-6
SWEShiftClicked : 0
SWETS : 1118939959734
SWEApplet : Account List Applet
-------------------------------------
CHILD PROPERTY SETS
-------------------------------------
(NONE)
-------------------------------------
END LEVEL : 0
-------------------------------------
```

Sample Code for Invoking Methods of Web Engine HTTP TXN Business Service

This topic contains the sample code for invoking the methods of the Web Engine HTTP TXN Business Service and writing the results to a text file. For more information, see "Example of Using Web Engine HTTP TXN Business Service" on page 111.

Add the following sample code to the WebApplet_InvokeMethod event:

```
function WebApplet_InvokeMethod (MethodName)
{
var fp = Clib.fopen('testfile.txt','a');
if ( fp == null )
{
TheApplication().RaiseErrorText(" ERROR Opening File ")
}
else
{
var oBS = TheApplication().GetService('Web Engine HTTP TXN');
var Inputs = TheApplication().NewPropertySet();
var Outputs = TheApplication().NewPropertySet();
var Headers = TheApplication().NewPropertySet();
var Cookies = TheApplication().NewPropertySet();
var tmpCookie = TheApplication().NewPropertySet();

Clib.fputs('====================================\n',fp);
Clib.fputs('WebApplet InvokeMethod event:\n',fp);
Clib.fputs('====================================\n',fp);

Clib.fputs('\n',fp);
Clib.fputs('====================================\n',fp);
Clib.fputs('Method: GetAllRequestCookies\n',fp);
Clib.fputs('====================================\n',fp);

Inputs.Reset();
Outputs.Reset();
oBS.InvokeMethod ( 'GetAllRequestCookies', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('====================================\n',fp);
Clib.fputs('Method: GetAllRequestHeaders\n',fp);
Clib.fputs('====================================\n',fp);

Inputs.Reset();
Outputs.Reset();
oBS.InvokeMethod ('GetAllRequestHeaders', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('====================================\n',fp);
Clib.fputs('Method: GetAllRequestParameters\n',fp);
Clib.fputs('====================================\n',fp);
```

```
Inputs.Reset();
Outputs.Reset();
oBS.InvokeMethod ('GetAllRequestParameters', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('====================================\n',fp);
Clib.fputs('Method: GetAllResponseCookies\n',fp);
Clib.fputs('====================================\n',fp);

Inputs.Reset();
Outputs.Reset();
oBS.InvokeMethod ('GetAllResponseCookies', Inputs, Outputs)
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('====================================\n',fp);
Clib.fputs('Method: GetAllResponseHeaders\n',fp);
Clib.fputs('====================================\n',fp);

Inputs.Reset();
Outputs.Reset();
oBS.InvokeMethod ('GetAllResponseHeaders', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('====================================\n',fp);
Clib.fputs('Method: GetAllServerVariables\n',fp);
Clib.fputs('====================================\n',fp);

Inputs.Reset();
Outputs.Reset();
oBS.InvokeMethod ('GetAllServerVariables', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('====================================\n',fp);
Clib.fputs('Method: GetRequestCookies\n',fp);
Clib.fputs('====================================\n',fp);

Inputs.Reset();
Outputs.Reset();

Inputs.SetProperty ('MY-COOKIE', '');
Inputs.SetProperty ('TestCookie', '');
Inputs.SetProperty ('Test1Cookie', '');

oBS.InvokeMethod ('GetRequestCookies', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('====================================\n',fp);
Clib.fputs('Method: GetRequestHeaders\n',fp);
Clib.fputs('====================================\n',fp);
```

```
Inputs.Reset();
Outputs.Reset();

Inputs.SetProperty ('MyHEADER', '');
Inputs.SetProperty ('MY_TEST', '');
Inputs.SetProperty ('CONTENT-TYPE', '');
Inputs.SetProperty ('CONTENT-LENGTH', '');

oBS.InvokeMethod ('GetRequestHeaders', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('====================================\n',fp);
Clib.fputs('Method: GetRequestInfo\n',fp);
Clib.fputs('====================================\n',fp);

Inputs.Reset();
Outputs.Reset();

oBS.InvokeMethod ('GetRequestInfo', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('====================================\n',fp);
Clib.fputs('Method: GetRequestParameters\n',fp);
Clib.fputs('====================================\n',fp);

Inputs.Reset();
Outputs.Reset();

Inputs.SetProperty ('TestQstr', '');
Inputs.SetProperty ('SWEActiveView', '');
Inputs.SetProperty ('SWECmd', '');
Inputs.SetProperty ('SWEMethod', '');
Inputs.SetProperty ('TestParam', '');

oBS.InvokeMethod ('GetRequestParameters', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('====================================\n',fp);
Clib.fputs('Method: GetResponseCookies\n',fp);
Clib.fputs('====================================\n',fp);

Inputs.Reset();
Outputs.Reset();

Inputs.SetProperty ('My-Test-COOKIE', '');
Inputs.SetProperty ('_sn', '');

oBS.InvokeMethod ('GetResponseCookies', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);
```

```
Clib.fputs('\n',fp);
Clib.fputs('====================================\n',fp);
Clib.fputs('Method: GetResponseHeaders\n',fp);
Clib.fputs('====================================\n',fp);

Inputs.Reset();
Outputs.Reset();

Inputs.SetProperty ('Content-Language', '');
Inputs.SetProperty ('MyHeader', '');

oBS.InvokeMethod ('GetResponseHeaders', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('====================================\n',fp);
Clib.fputs('Method: GetResponseInfo\n',fp);
Clib.fputs('====================================\n',fp);

Inputs.Reset();
Outputs.Reset();

oBS.InvokeMethod ('GetResponseInfo', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('====================================\n',fp);
Clib.fputs('Method: GetServerVariables\n',fp);
Clib.fputs('====================================\n',fp);

Inputs.Reset();
Outputs.Reset();

Inputs.SetProperty ('AUTH-USER-ID', '');
Inputs.SetProperty ('SERVER-NAME', '');

oBS.InvokeMethod ('GetServerVariables', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('====================================\n',fp);
Clib.fputs('Method: GetWebSessionInfo\n',fp);
Clib.fputs('====================================\n',fp);

Inputs.Reset();
Outputs.Reset();

oBS.InvokeMethod ('GetWebSessionInfo', Inputs, Outputs);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('====================================\n',fp);
Clib.fputs('Method: SetResponseCookies\n',fp);
Clib.fputs('====================================\n',fp);
```

```
Inputs.Reset();
Outputs.Reset();

tmpCookie = null;
tmpCookie = TheApplication().NewPropertySet();

tmpCookie.SetType ('My_Test_Cookie');
tmpCookie.SetValue ('Cookie Value for My_Test_Cookie');
tmpCookie.SetProperty ('Max-Age', '23434343');
tmpCookie.SetProperty ('Domain', '.example.com');
tmpCookie.SetProperty ( 'Path', 'eapps/test/cookie/path');

Inputs.AddChild (tmpCookie);

tmpCookie = null;
tmpCookie = TheApplication().NewPropertySet();

tmpCookie.SetType ('Another_Cookie');
tmpCookie.SetValue ('Cookie Value for Another_Cookie');
tmpCookie.SetProperty ('Max-Age', '23434343');
tmpCookie.SetProperty ('Domain', 'esales.example.com');
tmpCookie.SetProperty ('Path', 'esales/cookie/path');

Inputs.AddChild (tmpCookie);

oBS.InvokeMethod ('SetResponseCookies', Inputs, Outputs);
Clib.fputs('-----------------------------------\n',fp);
Clib.fputs('Input Cookies\n',fp);
Clib.fputs('-----------------------------------\n',fp);
WebApplet_OutputPropertySet(Inputs, 0, fp);

oBS.InvokeMethod ('GetAllResponseCookies', Inputs, Outputs);
Clib.fputs('-----------------------------------\n',fp);
Clib.fputs('Output Cookies\n',fp);
Clib.fputs('-----------------------------------\n',fp);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('===================================\n',fp);
Clib.fputs('Method: SetResponseHeaders\n',fp);
Clib.fputs('===================================\n',fp);

Inputs.Reset();
Outputs.Reset();

Inputs.SetProperty ('MyHeader', 'THIS is MyHeader');

oBS.InvokeMethod ('SetResponseHeaders', Inputs, Outputs);
Clib.fputs('-----------------------------------\n',fp);
Clib.fputs('Input Headers\n',fp);
Clib.fputs('-----------------------------------\n',fp);
WebApplet_OutputPropertySet(Inputs, 0, fp)
```

```
oBS.InvokeMethod ('GetAllResponseHeaders', Inputs, Outputs);
Clib.fputs('-----------------------------------\n',fp);
Clib.fputs('Output Headers\n',fp);
Clib.fputs('-----------------------------------\n',fp);
WebApplet_OutputPropertySet(Outputs, 0, fp);

Clib.fputs('\n',fp);
Clib.fputs('===================================\n',fp);
Clib.fputs('Method: SetResponseInfo\n',fp);
Clib.fputs('===================================\n',fp);

Inputs.Reset();
Outputs.Reset();
Headers.Reset();
Cookies.Reset();

Headers.SetType ('HEADERS');
Headers.SetProperty ('ABC_RESPONSE_HEADER1', 'RESPONSE_HEADER1 Value');
Headers.SetProperty ('ABC_RESPONSE_HEADER2', 'RESPONSE_HEADER2 Value');
Headers.SetProperty ('ABC_RESPONSE_HEADER3', 'RESPONSE_HEADER3 Value');
Headers.SetProperty ('ABC_RESPONSE_HEADER4', 'RESPONSE_HEADER4 Value');
Inputs.AddChild( Headers);

Cookies.SetType('COOKIES');

tmpCookie = null;
tmpCookie = TheApplication().NewPropertySet();

tmpCookie.SetType ('My_Test_Cookie2');
tmpCookie.SetValue ( 'Cookie Value for My_Test_Cookie2');
tmpCookie.SetProperty ( 'Max-Age', '23434343');

Cookies.AddChild (tmpCookie);

tmpCookie = null;
tmpCookie = TheApplication().NewPropertySet();

tmpCookie.SetType ('Another_Cookie2');
tmpCookie.SetValue ('Cookie Value for Another_Cookie2');
tmpCookie.SetProperty ('Max-Age', '23434343');

Cookies.AddChild (tmpCookie);

Inputs.AddChild (Cookies);

oBS.InvokeMethod ('SetResponseInfo', Inputs, Outputs);
Clib.fputs('-----------------------------------\n',fp);
Clib.fputs('Input Info\n',fp);
Clib.fputs('-----------------------------------\n',fp);
WebApplet_OutputPropertySet(Inputs, 0, fp);

oBS.InvokeMethod ('GetResponseInfo', Inputs, Outputs);
Clib.fputs('-----------------------------------\n',fp);
Clib.fputs('Output Info\n',fp);
Clib.fputs('-----------------------------------\n',fp);
WebApplet_OutputPropertySet(Outputs, 0, fp);
```

```
Clib.fclose(fp);
}
}
```

Index

Q

R

S