SIEBEL 7
eBusiness

SIEBEL INTERACTIVE DESIGNER API REFERENCE

VERSION 7.5, REV. A

12-DX88XR

NOVEMBER 2002

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

Printed in the United States of America

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

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

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

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

Proprietary Information

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

Contents

Chapter 4. Utility Functions

Chapter 5. User Interface Layout and Control Functions

Chapter 6. Pageset Functions

Chapter 7. Contents List Functions

Chapter 8. Callout Point Functions

Chapter 9. Data Access Functions

Chapter 10. Data Objects

Chapter 11. API Examples

Index

Introduction

This guide describes how to use the Interactive Designer API to customize eAdvisor and browser-based eConfigurator applications. These applications are intended for users who need to configure products, to create quotes, or to purchase products from Siebel customer and partner application Web sites.

Although job titles and duties at your company may differ from those listed below, the audience for this guide consists of users in the following category:

Model Developers Those responsible for designing and maintaining product configuration models.

All users should have detailed knowledge of their company's products and how they are sold. Before using Interactive Designer, you should also be familiar with the Siebel eBusiness Applications user interface and its basic operations. You should know how to accomplish basic tasks such as navigating between screens and views, selecting from lists, and entering information.

If you will be publishing configuration models on the Web, you should also be familiar with Siebel customer and partner applications, particularly the catalog views in Siebel eSales. For more information on how Interactive Designer fits into Siebel Interactive Selling, and additional applications you may need to understand, see the Overview chapter in *Siebel Interactive Designer Administration Guide*.

How This Guide Is Organized

This guide groups similar functions into chapters, so you can reference the API you need. Use this guide in conjunction with *Siebel Interactive Designer Administration Guide*.

For users creating stand-alone applications, read all of this guide except for Chapter 3, "Siebel-Specific Functions."

For users creating Siebel-integrated applications, read all of this guide.

Revision History

Siebel Interactive Designer API Reference, Version 7.5, Rev. A

November 2002 Bookshelf

Book Version: Rev A

Topic	Revision
"APP_PRICE_DATA" on page 20	Corrected default values.
"intl.js" on page 25	Corrected method name from intl to _SWEmsgAry in method for message content definition.
"Loading and Processing Messages" on page 26	Added this subsection, which describes how to tailor the messages that users see when Siebel eAdvisor is loading the application or is processing an input request.
"AddToSSCart" on page 46	Added examples for this function.
"GetPrice" on page 58	Updated description.

Topic	Revision
"Using the CDA Service Broker in the ISSCDA RT UI Service" on page 65	Added this subsection, which describes how to call the CDAServiceBroker method in the ISSCDA RT UI Service to have the client receive a confirmation response.
"ShowProductDetails" on page 71	Updated the description.
"FrameToOLString" on page 76	Corrected function name.
"Example 5: Use the CDA Service Broker" on page 181	Added this section.

1 Application File Reference

This chapter describes the Interactive Designer files that make up an application. It also describes the Siebel application file in which you can set variables that affect your application.

Setting Variables in the Siebel Application CFG File

In the Siebel application .cfg file (for example, uagent.cfg if you are using Call Center), you can set up your application to reference other Siebel data by setting the variables shown in Table 1.

You can also configure these properties in Siebel Tools. When you create an applet in Siebel Tools, you can specify what CFG properties to pass along to the ShowCDA function. For more information, see the section about invoking the ShowCDA Method from a button in *Siebel Interactive Designer Administration Guide*.

Table 1. Siebel Application CFG File Variables

Variable	Description
ISSCDAAutoDeployment	Used in Mobile mode, this flag enables/disables automatic projects deployment. If this value is TRUE, it will deploy the project files to your local file system. If it is FALSE, it will turn off automatic projects deployment. The default value is TRUE.
ISSCDADeploymentMode	Used in conjunction with ISSCDAAutoDeployment, this flag sets the automatic deployment mode. If the value is set to "ONDEMAND," only the projects that are being referenced by the browser-based engine are deployed to your local file system. If the value is set to "ALL," all projects that can be assessed by a user will be deployed to your local file system when the browser-based engine tries to load the first project.
ISSCDAGetMyPriceFields	The field names to be output to the client side in the GetPrice function.
ISSCDAHeaderBusCompName	The business Component name for the shopping cart. The default value is Quote.
ISSCDAHeaderBusObjName	The business Object name for the shopping cart. For example, Quote or Catalog. The default value is Catalog.
ISSCDAHeaderViewName	The view name the AddToCart function will switch to. This view name is also used after exiting GotoConfigurator.

Table 1. Siebel Application CFG File Variables

Variable	Description
ISSCDAIntegrationObjName	The integration object name to be used by EAI to query/sync/upsert to the database. The default value is Quote.
ISSCDAListItemBusCompName	The Business component at the item level. The default value is Quote Item.
ISSCDAProdDetBusCompName	The Business Component Name that a product detail view will use.
ISSCDAProdDetBusObjName	The Business Object Name that a product detail view will use.
ISSCDAProdDetViewName	The product detail view name.
ISSCDASSConfiguratorViewName	This view name the server-based eConfigurator will display.

Files in the Application Base Directory

This section describes the files in the application base directory.

home.htm

This is the home page of the application and it is generated by Interactive Designer based on the application template you use for the Interactive Designer project. The application loads and runs when you open this file in a browser. Edit the default content of this page to match the look and feel of the rest of your application.

kernel.htm

This file is part of the engine code set. Do not edit this file.

onl_boot.htm

This file specifies which browsers the application supports and defines alert messages that appear when a user opens the application in an unsupported browser. The home page of a application must reference this file.

Siebhome.htm

This file specifies the default project. Do not edit this file.

About the cs Directory

The cs directory contains the engine code for the application. Do not edit files in this directory.

Files in the Custom Directory

The custom directory contains JavaScript files that you can edit to customize the behavior of the applications' engine without modifying the core engine code itself.

app_config.js

The Application Configuration file contains variables that define properties of the application. The file contains all required variables, set to their default values. You can add optional variables to further customize the behavior of the application.

For variables that accept strings, all strings must be inside quotation marks.

Table 2 lists the variables that must be included in the Application Configuration file.

Table 2. Required Variables in the Application Configuration File

Variable	Value	Description
APP_DATA_VERSION	string	Defines a version number for the feature and configuration data in your application. After a user clicks a link from an order back to the application, the engine reads this information to check data versions.
APP_AUTO_LOAD_RESULTS	boolean	Enables automatic loading of output targets when a pageset first loads.
APP_RELOAD_ALL	boolean	Defines whether all display pages reload when output targets are generated.
APP_RELOAD_INPUTS_ON_EXC	boolean	Defines whether all display pages that contain input UI controls reload as an exception message displays, after a user selects an invalid configuration.
APP_DEFAULT_TIMEOUT	integer	Specifies the amount of time (in milliseconds) the browser waits for a file to load before it times out. Without this variable, the default time-out length is 5000 milliseconds.

Table 2. Required Variables in the Application Configuration File

Variable	Value	Description
APP_ABOUT_WIN_ARGS	string	Sets properties of the window into which the About file loads. Use the same properties inside the string (such as "scrollbars = 1, resizable = 0, width = 450") that you would use to launch a separate browser window from inside any HTML file.
APP_EXC_DISPLAY_NUM	integer	Defines the maximum number of exception messages that should appear.
APP_HELP_WIN_ARGS	string	Sets properties of the window into which the Help file set loads. Use the same properties inside the string (such as "scrollbars = 1, resizable = 0, width = 450") that you would use to launch a separate browser window from inside any HTML file.
TRANSACT_URL	string	Defines the URL to a Transact Server™. See *Siebel Interactive Selling Transact Server Interface Reference*.
TRANSACT_THIRD_PARTY_CART	boolean	Defines whether the Transact Server uses a third-party shopping cart. See *Siebel Interactive Selling Transact Server Interface Reference*.
TRANSACT_CART_WINARGS	string	Sets properties of the window into which the Transact Server loads a shopping cart. Use the same properties inside the string (such as "scrollbars = 1, resizable = 0, width = 450") that you would use to launch a separate browser window from inside any HTML file. See *Siebel Interactive Selling Transact Server Interface Reference*.
TRANSACT_SHOW_CART_URL	string	Defines the URL to a third-party shopping cart. Use the Transact ShowCart function to link from the application to the shopping cart located at this URL. See *Siebel Interactive Selling Transact Server Interface Reference*.
TRANSACT_ACTIVE	boolean	Determines whether the application runs in conjunction with Transact Server. See *Siebel Interactive Selling Transact Server Interface Reference*.
ORDER_SUBVAR	string	Specifies a string whose elements initialize the variables that track subitem subtotal and order total. See *Siebel Interactive Selling Transact Server Interface Reference*.

The following four required configuration variables allow the engine to access module configuration variables. Do not edit the content or location of these variables.

```
APP_ENGINE_CHANGED

APP_CONFIG_LOADED

ORDER_CONFIG_LOADED

TRANSACT_CONFIG_LOADED
```

These variables should appear at the end of the Application Configuration file. Do not add any variables after them.

Table 3 lists optional variables you can add to the Application Configuration file to further customize the appearance and behavior of the application.

Do not add any of these variables to the file after the *_CONFIG_LOADED variables.

Table 3. Optional Variables in the Application Configuration File

Variable	Value	Description
APP_ALWAYS_KEEP_BACK_STATE	boolean	If you work with large pagesets, you can set this variable to “false” in order to improve performance speed. By default, this variable is set to “false.”
APP_DISPLAY_AREA_FRAME	string	Defines the location of the display area. Use the syntax, APP_DISPLAY_AREA_FRAME = ISSStr + ".displayArea" If top.swe.contentFrame is defined, it is used instead, overriding the value in the config variable.
APP_LOAD_UI_ON_STARTUP	boolean	Shows the UI when the engine loads. If this variable is set to false, the persistent engine is used.
APP_SIEBEL_INTEGRATION_ON	boolean	Determines whether or not to include Siebel integration code.
APP_VERSION	string	Defines a version number for the application.
APP_HELP_URL	string	Specifies a URL that overrides the default Help page location.
APP_ABOUT_URL	string	Specifies a URL that overrides the default About page location.

Table 3. Optional Variables in the Application Configuration File

Variable	Value	Description
APP_SHOW_DATA_LOADING_PAGE	boolean	Specifies whether a message appears inside the application while the application is loading a pageset. If you do not add this variable, a message appears by default.
APP_SHOW_CALC_PAGE	boolean	Specifies whether a processing message appears in the exception message area. This occurs after the user makes a selection in an input UI control but before the associated output target appears. If you do not add this variable, a message appears by default.
APP_PRECONFIG_SEP_CHAR	string	Defines the character used to separate items in a list of dynamic default (DYNDEF) strings. The default character is a comma.
APP_PRECONFIG_EQUALS_CHAR	string	Defines the character used to equate a key with a value in a dynamic default (DYNDEF) string. The default character is an equal sign.
APP_PRICE_DATA	string	Defines what appears in the Pricing window. The strings you specify in this variable need to match what is being sent from the server. For example, the server sends an object that contains the attributes 'price,' 'description,' ‘quantity,’ ‘color,’ and ‘size.’ The exact data sent is determined by the server. To display the description, quantity, and price, set the variable using the following line: “var APP_PRICE_DATA = new Array(‘description’, ‘quantity’, ‘price’); Default value: 'Product Name,' 'Net Price,' 'Start Price,' 'Pricing Comments'
APP_PRICE_TITLE_ATTR	string	Defines the title for the pricing window. Default value: “Pricing Window”
APP_PRICE_BODY_ATTR	string	Defines the behavior of the body tag in the Pricing window, including background color, the onLoad event, or any of the valid body attributes or events. Default value: “bgcolor = #ffffff”
APP_PRICE_FONT_ATTR	string	Determines the fonts displayed in the Pricing window. Default value: “face = ’Verdana, Arial’ size = 2 color = ‘blue’”

Table 3. Optional Variables in the Application Configuration File

Variable	Value	Description
APP_PRICE_TABLE_ATTR	string	Use this variable to define the Pricing table attributes. All pricing information appears in a table. Default value: "border = 0 cellpadding = 2 cellspacing = 2 width = 100%"
APP_PRICE_CLOSE_ATTR	string	Specifies the text to display for the Close Window link for the Pricing window. For AOL users, it may become difficult to keep track of newly opened windows, so a Close Window link appears in the Pricing window. Default value: Close Window
APP_PRICE_WIN_ATTR	string	Defines attributes of the Pricing window. Using this variable, you can adjust the size (height, width), the controls displayed, whether or not the window is resizable, and any other valid window attributes. This variable is particularly useful for sizing the window to the data that is expected to appear. Default value: "status = 0,scrollbars = 1,resizable = 1,width = 450,height = 200"
APP_SOA_TOP_LOC	string	Defines location of frame containing the application. Use this variable when Start on Active module is being used and the application is in a nested frameset. Default: top
TRANSACT_CART_TARGET	string	Determines the type of window in the Transact Server loads. The type of window is defined in Javascript.
TRANSACT_NOT_ACTIVE_MSSG	string	Specifies the text content of the message that appears when users try to interact with an unavailable Transact Server.
TRANSACT_OPEN_QUOTE_PROMPT	string	Specifies the text that precedes display of an open quote from the Transact Server.

Optional Variables for Use with Session Timeout Problems

Use the following optional variables if you are having problems with session timeout. Using these variables, as shown in Table 4, puts you in session simulator mode. The session simulator will ping the server at an interval equal to half the session length, as long as any qualifying client-side activity has occurred within that time frame.

Table 4. Session Timeout Variables

Variable	Value	Description
APP_SESSION_LENGTH	integer	Set this variable to the length of the Siebel server session in minutes. Default = 15.0.
APP_PING_SERVER_URL	string	Specifies a URL to be "pinged." A default values is retrieved using ISS.GetTopURL().

customCode.js

This is the file inside which you can write custom code to be associated with a callout point function.

Sample usage:

```
function COP_InvalidItemAdded(calloutPkg)
{
alert("Behold the invalid configuration.");
}
```

The customCode.js file also contains the InitApp function, which determines the behavior of the application when it first loads. You can write custom code to define the behavior. You can also edit either of the included InitApp examples to display the Contents List or a specific pageset when the application loads. You can even use InitApp to return a string if you want to override the default module registry file. InitApp can also be used to activate Start On Active when using an application in a stand-alone environment.

NOTE: If using the persistent engine (APP_LOAD_UI_ON_STARTUP = false), InitApp should not contain code that displays anything in the display area. This is because the display area is not available until you call ShowCDA().

Sample Usage:

```
function InitApp() {

ISS.ShowContentsList();

}
```

Files in the ds Directory

The ds directory contains the Pageset Properties file, pagesetID_x.js, as well as product data files generated from the information you enter in Interactive Designer Configuration and Feature tables.

pagesetID_x.js

Pageset Properties File

This file contains functions that define various pageset properties. Usually, you define these properties inside Interactive Designer, which writes them to this file when you generate application files from the project. This file must not contain any HTML tags.

The file should start with the StartXInfo(*pagesetID*) function and end with the EndXInfo(*pagesetID*) function. In between, the file can contain any of the following functions:

- InitPagesetVersion(version_string)

  Defines a version number string for the pageset. After a user clicks a link from an order back to the application, the engine reads this information when checking data versions.

- InitPagesetDesc(description_string)

  Defines a text description for the pageset.

- InitPagesetItemized(FALSE/TRUE)

  Specifies whether the components of a product appear as subitems or main line items after they are passed to an order-generating tool. To display product components as subitems, set the variable to FALSE (the default). To display product components as main line items, set the variable to TRUE.

- InitAltOMSUrl(URL)

  Specifies the URLs of additional order management systems to which configurations generated by the pageset are sent. This is useful if you are using a Transact Server with the application and need to send configuration information to more than one order management system. See *Siebel Interactive Selling Transact Server Interface Reference*.

prodlistdata.htm

This file is generated by Interactive Designer, and it contains the data used to define Contents Lists in the application.

Do not edit this file directly. If you need to update contents list data, edit the contents list table inside your Interactive Designer project and redeploy your project to recreate this file.

Files in the jd Directory

The jd directory contains the application module registry and its associated files. You will not need to edit the files in this directory often, but if you plan to deploy your application in multiple languages, you might want to customize text contained inside the intl.js file.

intl.js

This file contains the text strings used in alert messages. You can edit these strings to internationalize your application.

Additionally, if your application contains custom code that requires text strings, you should define the strings inside this file.

To define a text string inside the intl.js file

1. Define a variable for the string.
2. Define the message content of the text string.

NOTE: All text strings must be in double quotes. You can use arguments inside the strings.

To call the text string you defined, use the following syntax inside your custom code:

```
ISS.ErrIntern(STRINGNAME[, arg1, arg2, ... arg_n])
```

where:

STRINGNAME Name (variable) used to identify the text string inside the intl.js file

arg_n Optional arguments

Sample usage:

Define the string variable:

```
var TESTMSG=0;
```

Define text message content:

```
_SWEmsgAry[“TESTMSG”]= "%1 Fish, %2 Fish,  %3 Fish, %4 Fish";
```

Inside your code, call the text string:

```
ISS.ErrIntern(ISS.TESTMSG,"one","two","red","blue");
```

Loading and Processing Messages

You can optionally tailor the messages that users see when Siebel eAdvisor is loading the application or is processing an input request. To do this:

1. Define the message content of one or both of the variables listed in Table 5.
2. Modify the message arrays in the intl.js file.

Table 5. Variables that customize user messages

Variable	Value	Description
ISSCDA_DEFAULT_DATA_LOAD_STR	string	Specifies the text content of the message that appears inside the application while the application is loading a pageset. If this variable is not defined, the default message “Loading...” is used.
ISSCDA_DEFAULT_CALC_PAGE_STR	string	Specifies the text content of the processing message that appears in the exception message area. This occurs after the user makes a selection in an input UI control but before the associated output target appears. If this variable is not defined, the default message “Calculating...” is used.

Sample usage:

Open the intl.js file with a text editor and define one or both variables as follows:

```
_SWEmsgAry["ISSCDA_DEFAULT_DATA_LOAD_STR"] = "Data Loading...";
_SWEmsgAry["ISSCDA_DEFAULT_CALC_PAGE_STR"] = "Calculating...";
```

Replace the text string within quotes on the right side of the equal (=) sign with the text of your choice.

Files in the pg Directory

The pg directory contains the Pageset UI Registry files, the Pageset UI Definition files, and the display pages for all the pagesets in your application, as well as the cascading style sheet that defines the appearance of the Contents List.

You should also use this directory to store any additional HTML and image files you use to customize the appearance of display pages.

pagesetID_1.htm . . . pagesetID_"n".htm

Display Pages

These are HTML files that display input UI controls and output targets to help users choose and configure a product that meets their needs.

Display pages appear inside the frameset layout defined by the Pageset UI Definition file. This specific layout in turn appears inside a smaller area of the larger application. This specific layout is defined in the Application UI Definition file.

Interactive Designer generates default display pages for each pageset. The file names of the default display pages begin with the pageset ID and are numbered sequentially. If you want to rename the display pages, you must reassociate the new files names with pageset frames inside the Pageset UI Registry file.

Most applications place input UI controls and output targets on separate display pages, but you can place input UI controls and output targets on the same page. Each pageset must contain at least one display page.

NOTE: Interactive Designer also provides the Input UI tab on which you can add UI controls without writing code. See *Siebel Interactive Designer Administration Guide* for information.

To add an input UI control to a display page

1. Open the HTML source file for the display page.
2. In the location where you want the input UI control to appear, add the following:

```
<SCRIPT>
```

```
document.write(ISS.BuildWidget(type,window,
name,[int1,int2,Prefill]));

</SCRIPT>
```

where:

type	Specifies the type of input UI control to create. Valid arguments are CHECKBOX, GETTEXT, LISTBOX, RADIO, and MAP.
name	Defines the source of the content that populates the input UI control. For check box, list box, and radio button input UI controls, this is the name of a Feature table in the Interactive Designer project. For text entry input UI controls, this is the name you want to give to the text entry field. The application stores user input to this field under the name you use here.
int1	For list box input UI controls, this argument is an integer that specifies the number of visible rows for the list box. Set this argument to 1 to create a drop-down list. For radio button input UI controls, this argument is an integer that specifies which Feature table row is associated with the particular radio button. To create a group of radio buttons, you must create one input UI control for each radio button feature value. For text entry input UI controls, this argument is an integer that specifies the width of the text entry field, as defined by the number of characters visible in the default browser font. Check box input UI controls do not take any arguments.
int2	For list box input UI controls only, this argument is an integer that specifies the width of the list box, as defined by the number of characters visible in the default browser font.
prefill	For list box input UI controls only, this boolean value sets the width of the list box control to accommodate the widest item in the list. If prefill is TRUE and Width is greater than the number of characters in the longest line of text appearing in the list box, the list box appears with a width equivalent to Width. If prefill is FALSE, the width is set to accommodate the longest line of text that appears in the list box.

If necessary, use HTML just outside the input UI control definition to write a caption for the input UI control.

The following sample code creates a list box input UI control whose selections are defined by the values in a Feature table called SIZE. This list box input UI control appears as a drop-down list, as specified by the 1 argument, that is approximately 50 characters wide, as specified by the 50 argument. The list box input UI control is preceded by a caption, Size:, defined with regular HTML.

```
<FONT face=verdana size=1><B>Size:</B></FONT>

<SCRIPT>

document.write(ISS.BuildWidget("LISTBOX",window,"SIZE",
1,50,true));

</SCRIPT>
```

NOTE: Interactive Designer also provides the Output UI tab on which you can add UI controls without writing code. See *Siebel Interactive Designer Administration Guide* for information.

To add an output target to a display page

1. Open the HTML source file for the display page.
2. In the location where you want the output target to appear, add the following:

```
<SCRIPT>

document.write(ISS.BuildTarget(type,window,
name[,arg1,arg2]));

/SCRIPT>
```

```
where:
```

type

Specifies the type of output target to create. Valid arguments are LINK, PICT, TEXT, SUBCONFIG_LINK, OPT_SUBCONFIG_LINK, NB_LINK, and NB_RETURN.

name

Defines the source of the content that populates the output target, which can be any of the following:

- A column in a Configuration table. In this case, the argument is simply the name of the column.
- A column in a Feature table. In this case, the argument is the name of the Feature table and the name of the specific column inside of it, separated by a period, as in COLOR.DESC.
- The Feature table that populates an input UI control from which the user has made a selection. In this case, the argument is simply the name of the Feature table.
- The name of a text entry input UI control in which text is typed and stored. In this case, the argument is the name you used to define the text entry input UI control.

arg1

For image (PICT) output targets, this optional argument is an integer that defines the width of the image, in pixels.

For link output targets, this argument is a Boolean value that determines whether data in the column identified by the *name* argument is a pageset (TRUE) or a URL (FALSE). If the data is a URL and you set the argument to FALSE, the URL loads into a separate window, instead of inside the main application, when you click the link output target.

For image (PICT) output targets, this optional argument is an integer that defines the height of the image, in pixels.

For link output targets, this optional argument is a string that specifies the name of a table column (or columns, separated by commas) in which a preconfigured condition is defined for the pageset.

This optional argument is a string that specifies the name of a table column (or columns, separated by commas) in which a preconfigured condition is defined for the pageset.

When you click a link target containing a preconfigured condition, the application overrides the default settings of input UI controls in the pageset with the settings defined in the preconfigured condition.

The definition of a preconfigured condition must contain the name of the overriding Feature table and the feature (CODE) value of the desired override setting.

If you define a preconfigured condition column in a Configuration table, the argument references only the name of the Configuration table column whose values override the defaults.

If you define a preconfigured column in a Feature table, the argument must contain the name of the Feature table *and* the name of the specific column inside of it whose values override the defaults, separated by a period (as in COLOR.PRECONFIG).

arg2

If necessary, use HTML just outside the input UI control definition to write a caption for the input UI control.

Sample usage:

The following sample code creates a text output target whose text content is defined inside the DESC column of a Feature table named SIZE, as specified by the SIZE.DESC argument. The text output target is preceded by a caption, Size:, defined with regular HTML. HTML formatting also determines that the font of the text output target, like that of its caption, is Verdana or Helvetica.

```
<FONT face="verdana,helvetica" size=2><B>Size:</B>

<SCRIPT>

document.write(ISS.BuildTarget("TEXT",window,
"SIZE.DESC"));

</SCRIPT>

</FONT>
```

pagesetID_i.htm

Pageset UI Registry File

Interactive Designer creates a Pageset UI Registry file for each pageset you define in an Interactive Designer project. The Pageset UI Registry file serves three purposes:

- Associates a Pageset UI Definition file with a particular pageset, using the RegisterFrameSet function
- Defines which display pages appear in each of the frames defined in the Pageset UI Definition file of the pageset, using the RegisterPageLocation function
- Specifies which of the frames defined in the Pageset UI Definition file of the pageset display exception messages, using the RegisterExceptionFrames function

The file should start with the StartUIInfo(pagesetID) function and end with the EndUIInfo(pagesetID) function inside the onLoad event handler of the opening <BODY> tag. The file must also reference the header.js file from inside the <HEAD> tags, as in:

```
<HEAD><SCRIPT src="../jd/header.js"></SCRIPT></HEAD>
```

Associating a Pageset UI Definition File with a Particular Pageset

Because each pageset can have its own frameset layout, you must specify which file defines the frameset layout of the current pageset. Usually the Pageset UI Definition file is named pagesetID_ui.htm.

Use the RegisterFrameSet function to identify the Pageset UI Definition file associated with the current pageset. The syntax of this function is:

```
RegisterFrameSet(name,frameset_name,frame_name[,path])
```

where:

name	String used as the setLoaded parameter of the Pageset UI Definition file. This is usually the name of the frameset.
frameset_name	String identifying the file name of the Pageset UI Definition file.

frame_name	String identifying the complete path, from the top of the application, and name of the application frame, by default, mainArea, inside which the pageset frameset appears.
path	Optional string identifying the complete path.

Sample usage:

```
ISS.RegisterFrameSet(“dogs_ui","dogs_ui.htm",
ISS.GetPagesetDisplayArea()
“server_name/CDA_application_root_directory/ui”);
```

The first argument of the RegisterFrameSet function. In the example above, dogs_ui must be identical to the string used as the setLoaded parameter, most commonly, oc_default_ui or pagesetID_ui, inside the Pageset UI Definition file.

Defining Which Frames Display Each of the Display Pages

Because your application can contain any number of display pages and any number of frames, you must tell the engine which display pages appear in each frame defined for the pageset.

Use the RegisterPageLocation function to identify the frame that each display page loads into. The syntax of this function is:

```
RegisterPageLocation(file_name, frame_name, visible[, path]);
```

where:

file_name	String identifying the file name of the display page being assigned to a frame
frame_name	String identifying the complete path, from the top of the application, to the frame into which the display page loads
visible	String that specifies whether the display page is visible, if the string value is TRUE, or invisible, if the string value is FALSE, when it first loads into the frame
path	Optional string identifying the complete path to the display page file name

Sample usage:

```
RegisterPageLocation("dogs.htm",
“ISS.GetPagesetDisplayArea() + .ui_disp”
```

Specifying Which Frame Displays Exception Messages

Because your application can contain any number of frames, you must tell the engine which frame displays exception messages defined for the pageset.

Use the RegisterExceptionFrames function to identify the frame that exception messages for the pageset load into. The syntax of this function is:

```
RegisterExceptionFrames(frame_name);
```

where:

frame_name String identifying the complete path, from the top of the application, to the frame in which exception messages appear.

Sample usage:

```
RegisterExceptionFrames
("ISS.GetPagesetDisplayArea() + .uidata”);
```

onlink.css

This is the cascading style sheet that defines the appearance of the contents list in the application.

oc_default_ui.htm

This file, also called the Pageset UI Definition file, defines the HTML frameset layout of a pageset. Specifically, the Pageset UI Definition file defines the nested frames, showing display pages, that appear inside a single frame, called mainArea in the application templates, of the application.

In other words, this file defines the frameset inside which display pages for a pageset appear. Associate display pages with these frames inside the Pageset UI Registry file. For example, if the Pageset UI Registry file contains the definition:

```
RegisterExceptionFrames
(ISS.GetPagesetDisplayArea() + “.uidata”);
```

the ol_ui (Application UI Definition) file must contain a frame called mainArea. The mainArea frame in turn must contain nested frames, including a frame called uidata into which exception messages load. The nested frames are defined inside the Pageset UI Definition file.

By default, Interactive Designer associates the default Pageset UI Definition file, oc_default_ui.htm, with all the pagesets in the Interactive Designer project. In this case, all display pages in your application appear inside the same frameset, regardless of which pageset they are part of.

If you want to change the frameset structure of the display page area based on which pageset the display pages belong to, you can create additional Pageset UI Definition files, *_ui.htm, that define unique framesets. Use the RegisterFrameSet function inside the Pageset UI Registry file to associate a Pageset UI Definition file with a particular pageset.

Use the SetContentsListFrame function inside a Pageset UI Definition file to identify which frame defined for the pageset displays the contents list. You can also use the RegisterContentsListFrame function inside the Application UI Definition file to display the Contents List as part of the main application layout, outside the display page area.

The Pageset UI Definition file must reference the header.js file from inside the <HEAD> tags, as in:

```
<HEAD><SCRIPT src="../jd/header.js"></SCRIPT></HEAD>
```

You must include a call to the SetLoaded function inside the onLoad event handler of the main frameset definition in the Pageset UI Definition file. Use the name of the Pageset UI Definition file without the extension, most likely oc_default_ui or pagesetID_ui, as the argument of the SetLoaded function.

pl_bullet1.gif, pl_bullet2.gif, pl_bullet3.gif, pl_bullet4.gif

These are images used in front of the first four levels of Contents List entries in the application.

Files in the ui Directory

The ui directory contains files that define the appearance of the application at its outermost level. Definitions in these files apply to the application in general and not to any specific pagesets that appear inside it.

This section lists the image and HTML files that are common to all applications. The ui directory of your own application may contain additional files that define various user interface components unique to your own application.

about.htm (example only)

The about.htm file contains generic content about applications that you can edit to address the specifics of your own application.

Use the ShowAbout function to link to the About file.

help*.gif (example only)

These are default images used inside the files that load into the Help file, helpset.htm.

help_*.htm (example only)

These files contain default help text and images that load into the frames defined by the Help file, helpset.htm.

helpset.htm (example only)

The helpset.htm file defines a frameset into which files containing generic help content, help_*.htm, load. You can edit the text of these files to address the specifics of your own application.

Use the ShowHelp function to link to the Help file.

nf_white.htm

This file defines a blank frame. To define a blank frame, use:

```
<frame src="javascript:''">
```

ol_fly.htm (example only)

This file provides an example of how the Interactive Designer APIs can be used to load the Contents List, Help file, and About file. The Flyover frame appears at the top of the browser window and persists throughout the entire application, regardless of which pageset is active.

ol_ui.htm

This file is also known as the Application UI Definition file. It defines the HTML frameset layout of the application. Each application contains a single Application UI Definition file.

The HTML frameset layout defined in the Application UI Definition file represents what a user sees when the application first loads, as well as the static areas surrounding the display pages as a user continues to interact with the application.

One of the frames defined in the Application UI Definition file usually outlines the area, called mainArea in the Interactive Designer templates, into which all the display pages in your application load. You must use the RegisterUI function to indicate which frame defines this pageset display page area. The nested frameset structure of this single area is managed by a Pageset UI Definition file.

Use the RegisterContentsListFrame function to identify which frame defined in the Application UI Definition file displays the contents list. You can also use the SetContentsListFrame function inside a Pageset UI Definition file to display the Contents List inside the frameset of a pageset.

The Application UI Definition file must reference the header.js file from inside the <HEAD> tags, as in:

```
<HEAD><SCRIPT src="../jd/header.js"></SCRIPT></HEAD>
```

You must include a call to the RegisterUI function inside the onLoad event handler of the main frameset definition in the Application UI Definition file. The RegisterUI function identifies which of the frames defined in this file contains the nested pageset framesets defined in Pageset UI Definition files.

trans.gif

This is an image that creates transparent vertical and horizontal space. Use width and height attributes within the Image tag to specify the size.

welcome.htm

This is a file that the application templates display inside the pageset UI definition area, called mainArea in the templates, when the application first loads.

Reserved Word Reference 2

This chapter lists the reserved words in Interactive Designer, browser-based applications, and JavaScript.

Interactive Designer and Browser-Based Applications Reserved Words

Table 6 lists the reserved words in Interactive Designer and browser-based applications like eAdvisor and eConfigurator.

Interactive Designer and browser-based applications use these reserved words to define specific areas of functionality, so you should not use these terms in circumstances not related to the functionality they represent. For instance, you can use the word RULE as a column name to define exceptions in an Interactive Designer configuration table, but you should not use it as the name of a feature table.

This rule applies to words used as names of Interactive Designer tables and table columns or as definitions of frames, variables, or functions in your browser-based application.

Table 6. Reserved Words in Interactive Designer and Browser-Based Applications

BNAME	FAMILY	RULE
BTYPE	FLYOVER	RELATIONSHIP
C	FULLPRICE	SINGLE
CATPAGE	INDEX	SINGLEINFO
CHILD	ITEM	SINGLEITEMS
CODE	ITEM_VARS	SUBITEMS
D	ITEMS	SUBITEMTOTAL
DEFAULT	LINEDISC	SUBPRICE
DESC	LINKBACKSTR	SUBPRICETOTAL
DESCR	MAINCLOSED	SUBTOTAL
DFT	MULTI	SUBVAR
DISCOUNT	MULTIINFO	TEMPLATES
DISCPRICE	ISS	TOP

Table 6. Reserved Words in Interactive Designer and Browser-Based Applications

DISCSUBPRICE	ONLINK	TOTAL
DISCTOTAL	ORDER_TYPE	TOTALVAR
DYNAOBJ	PAGE	UNITPRICE
DYNAWIN	PARENT	URL
DYNDEF	PRECONFIG_OBJ	VALID
EXTDESC	PRODSTR	WINARGS
EXTPRICE	QTY	
EXTSUBPRICE	QUANTITY	

JavaScript Reserved Words

Table 7 lists the reserved words in JavaScript. The reserved words in this list cannot be used as names of tables, columns, variables, functions, methods, or objects.

Table 7. Reserved Words in JavaScript

abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

Siebel-Specific Functions 3

Use Siebel-specific functions at various points in a browser-based application to integrate with other Siebel applications.

NOTE: The functions in this chapter cannot be used in stand-alone applications. Stand-alone users should skip this chapter.

AddToSSCart

Usage Use the AddToSSCart function to add the currently selected product to the order or quote. The product can be added without ending an application session.

AddtoSSCart can be called from any frame within the application and can be executed anywhere a JavaScript function can be used. It is commonly called from the onClick event handler of a link, but can also be referred to from an image map, set as the SUBMIT method of an embedded frame, or called from another user-defined JavaScript function.

Refer to the Referencing Other Siebel Data chapter in *Siebel Interactive Designer Administration Guide*.

Syntax AddToSSCart(*productDescriptionString, StayIneAdvisor*)

Argument	Description
productDescriptionString	Optional argument used as the description for the customizable product to be added. If productDescriptionString is not specified, the model and state are examined to automatically build a customizable product description before adding the product to, and displaying, the quote or order. You can define ProductDescriptionString in the following ways: ■ As < product ID > ■ As a fully formed string, built by hand or using helper APIs (see "BuildProductStr" on page 51). Use the form: `relationship=<relationship name>&*prodID=<prodID>&*qty=<quantity>&*attributes={<key1>=<value1>&*<key2>=<value2>&*...}&*children={<child1>|*<child2>|*...}` ■ As null to specify not to extract a string
StayIneAdvisor	Optional argument. The possible values are true or false. If set to true, when a link is clicked, users will not leave the eAdvisor session. Instead, users will get a javascript pop-up message indicating success or failure of adding the product to the cart. If set to false, when a link is clicked, users go directly to the shopping cart. AddToSSCart can be called with no parameters, but in order to explicitly set the behavior of *StayIneAdvisor*, you must have a value in the first parameter, *productDescriptionString*.

Examples

```
ISS.AddToSSCart();

ISS.AddToSSCart(ISS.GetBusCompId(“PROD.DESC”))

ISS.AddToSSCart("relationship=null&*prodID=
XFS-628&*qty=1&*attributes={}&*children={}");

ISS.AddToSSCart(ISS.GetBusCompId(“PROD.DESC”, true));

ISS.AddToSSCart(ISS.GetResultsValue("CONSTRUCTED_PROD"));

ISS.AddToSSCart(ISS.GetBusCompId("STEREO.PART_DESC"), false);

ISS.AddToSSCart(ISS.GetBusCompId(null, true));

ISS.AddToSSCart(null, true);
```

BuildAttributeList

Usage BuildAttributeList takes 1..n arguments of attributes strings and creates an array of them, which can be used for BuildProductStr.

Syntax ISS.BuildAttributeList(*AttrStr[] attrs*)

Argument	Description
attrs	1..n arguments of attribute strings.

BuildChildList

Usage BuildChildList takes 1..n arguments of product strings and creates an array of them, which can be used for BuildProductStr.

Syntax ISS.BuildChildList(*prodStr*)

Argument	Description
prodStr	Product string describing the product. 1…n prodstrings. Use null, without quotes around it, to specify that no product string should be sent. Use "auto" to automatically extract the product string. prodStr is added automatically when you choose to use it in the other arguments. Do not set it manually.

Example

```
ISS.BuildChildList(child1, child2, child3)
```

where childn is a prodstr

BuildProductStr

Usage

In cases where the customizable product string cannot be automatically generated from the model, construct the string programatically using BuildProductStr and the supporting cell functions documented in this section.

The function may be called in cell functions or as part of a button/link/javascript that executes after the configuration engine has run.

Syntax

ISS.BuildProductStr(*id, qty, AttrStr[] attrs, ProdStr[] children, relationshipName*)

Argument	Description
id	A string for the product ID.
qty	Optional argument. Integer value for the quantity. Defaults to 1.
attrs	Optional argument. Array of attribute strings.
children	Optional argument. Array of customizable product strings.
relationshipName	Optional argument. String for the relationship name defined in the creation of the customizable product.

Example

Example 1

```
var myChildStr = "relationship=ChildSlot&*prodID=ARS-625&*qty
=1&*attributes={}&*children={}";

ISS.BuildProductStr(“TRS-525”,1,ISS.BuildAttributeList(“Color
=Yellow”, “Texture=Coarse”), ISS.BuildChildList(myChildStr),
“Slot1”);

Returns:

"relationship=Slot1&*prodID=TRS-525&*qty=1&*attributes
={Color=Yellow&*Texture=Coarse}&*children
={relationship=ChildSlot&*prodID=ARS-625&*qty=1&*attributes
={}&*children={}}"
```

Example 2

```
var child1 =
ISS.BuildProductStr(ISS.GetBusCompID("CDPLAYER.PARTNUM"), 1, null,
null, "CD_PLAYER");

var child2Attrs =
ISS.BuildAttributeList(ISS.GetAttribute("SPEAKERS.WATT"));
```

```
var child2 =
ISS.BuildProductStr(ISS.GetBusCompID("SPEAKERS.PARTNUM"),4,
child2Attrs, null, "SPEAKERS");

var rootPart = ISS.BuildProductStr(ISS.GetBusCompID("ROOT"),1,
null, ISS.BuildChildList(child1, child2));

rootPart has value:

"relationship=null&*prodID=KJR-
157&*qty=1&*attributes={}&*children={relationship
=CD_PLAYER&*prodID=AFT-157&*qty=1&*attributes
={}&*children={}|*relationship=SPEAKERS&*prodID=HGT-697&*qty
=4&*attributes={Watts=60}&*children={}}"
```

Supporting Cell Functions

Table 8 lists the supporting cell functions to use when constructing a product string by hand.

Table 8. Supporting Cell Functions

Function	Description
ISS.AddAttributeToProductStr (ProdStr, newAttr)	Given a customizable product string and an attribute string, this function adds the attribute to the customizable product string, and returns the constructed string. The passed-in strings are unchanged.
ISS.AddChildToProductStr (ProdStr, newChild)	Given a customizable product string and a child customizable product string, this function adds the child string as a child of the customizable product string and returns the constructed string. The passed in strings are unchanged.
ISS.AddProdIDToProductStr (BuildProductStr, prodID)	Given a customizable product string and a product ID, this function changes the product ID of the customizable product string and returns the constructed string. The passed in strings are unchanged.
ISS.AddQuantityToProductSt (ProdStr, qty)	Given a customizable product string and a quantity, this function changes the quantity of the customizable product string and returns the constructed string. The passed in strings are unchanged.

Table 8. Supporting Cell Functions

Function	Description
ISS.AddRelationshipNameToProductStr (BuildProductStr, relationshipName)	Given a customizable product string and a relationship name, this function adds the relationship name to the customizable product string and returns the constructed string. The passed in strings are unchanged.
ISS.GetAttribute(col)	Given a column name, this function constructs an attribute string representing the attribute name and value associated with that column. Note that the value is dependent on the selected row, and this function should only be called from points after the engine has run. Example: `ISS.GetAttribute("COLOR.ATTR");` Returns: "Model color = Red"
GetBusCompID(col)	Given a column name, this function returns the value of the business component ID associated with that column. Note that the ID is dependent on the selected row, and this function should only be called from points after the engine has run. Example: `ISS.GetBusCompID("MODEL.DESC");` Returns: "L523"

BuildQuestionAnswerString()

Usage Use the BuildQuestionAnswerString() function to augment the creation of a sales opportunity in another Siebel application. This API returns a string of questions and answers corresponding to user selections.

Syntax BuildQuestionAnswerString(question1, answer1, question2, answer2)

The string is in the form: < question string > < answer string > , < question string > < answer string > .

Argument	Description
question1	Example question: “What is the industry?”
answer1	Example answer: “Automotive”

Example

```
ISS.BuildQuestionAnswerString("How many users?",
ISS.GetResultsValue("USERS.DESC"), "What is the industry",
ISS.GetResultsValue("INDUSTRY.SHORT_DESC"));
```

CreateOpportunity()

Usage Use the CreateOpportunity() function to create a sales opportunity in another Siebel application. This API sends information about the opportunity to the business service which creates a new sales opportunity.

Syntax CreateOpportunity(*name, prodID, questionAnswerString*)

Argument	Description
name	The name of the sales opportunity.
prodID	A single simple product ID for the product to be added to the opportunity.
questionAnswerString	The context string to be stored in a notes field. Often, the result of BuildQuestionAnswerString will be used.

Example

```
ISS.CreateOpportunity(ISS.GetResultsValue("ROOT_PROD"),
ISS.BuildQuestionAnswerString("How many users?",
ISS.GetResultsValue("USERS.DESC")));
```

GetCDAEntryArg()

Usage Use the GetCDAEntryArg() function to return a specified value in the argument array associated with the pageset name last passed into the ShowCDA function.

```
var myArgs = new Object();

myArgs [“userDiscount”] = 10;

ISS.ShowCDA(“Cars|sedans”, null, myArgs);
```

Within the application, the arguments sent can be accessed as follows:

```
var discount = ISS.GetCDAEntryArg(“userDiscount”);
```

discount will get the value 10.

GetCDAEntryArgs()

Usage Use the GetCDAEntryArgs() function to return the argument array last passed into the ShowCDA function. ShowCDA is called as follows:

```
var myArgs = new Object();

myArgs["userDiscount"] = 10;

ISS.ShowCDA("Cars|sedans", null, myArgs);
```

Within the application, the arguments sent can be accessed as follows:

```
var discount = args["userDiscount"];
```

discount will get the value 10.

GetPrice

Usage

Use the GetPrice function to return and display the final price of the selected product for the user. When the call is made, a pop-up window will display showing the current product price. The price will include any pricing adjustments if configured within the Siebel pricing engine.

GetPrice can be called from any frame within the application and can be executed anywhere a JavaScript function can be used. It is commonly called from the onClick event handler of a link, but can also be referred to from an image map, set as the SUBMIT method of an embedded frame, or called from another user-defined JavaScript function.

Refer to the Runtime Access to Your Pricing Information topic in *Siebel Interactive Designer Administration Guide* for more information.

Syntax

GetPrice(*productDescriptionString*)

Argument	Description
productDescriptionString	Optional argument used as the description for the customizable product to be added. If productDescriptionString is not specified, the model and state are examined to automatically build a customizable product description before adding the product to, and displaying, the quote or order. You can define ProductDescriptionString in the following ways: ■ As < product ID > ■ As a fully formed string, built by hand or using helper APIs (see "BuildProductStr" on page 51). Use the form: `relationship=<relationship name>&*prodID=<prodID>&*qty=<quantity>&*attributes={<key1>=<value1>&*<key2>=<value2>&*...}&*children={<child1>|*<child2>|*...}` ■ As null to specify not to extract a string

Example

```
ISS.GetPrice(ISS.GetBusCompID("CAR.COLOR"));

ISS.GetPrice();

ISS.GetPrice(ISS.GetResultsValue("CONSTRUCTED_PRODUCT"));
```

Additional Pricing Functions

You can modify the behavior of GetPrice by using the following functions. Call these functions from the customCode.js file if they are application specific. Call them from the _i file if they are page-level specific.

- ORP_DisplayPrice

  Allows you to write custom code to bypass the entire code. The user is passed the same information, including the price object, that is returned from the server and all the configuration variables.

- ORP_DisplayPriceString

  Allows you to write custom code to bypass the entire code. This function allows you to override the final display of the data. This function is useful if you want to override the default behavior of opening a pop-up window for displaying the price.

- COP_BeforeDisplayPriceString

  Allows you to manipulate the HTML string before it appears in the window.

For more information on these functions, see "Callout Point Functions" on page 137.

Accessing the Additional Pricing Functions

Use the following accessories to the price object to access the additional pricing functions:

- GetField (fieldName)

  Retrieves the value specified by fieldName.

- GetChildren ( )

  Returns an array of price objects or an empty array if there are no children.

GotoSSConfigurator

Usage

Use the GotoSSConfigurator function to hand off a product to the Siebel server-based eConfigurator.

GotoSSConfigurator can be called from any frame within the application and can be executed anywhere a JavaScript function can be used. It is commonly called from the onClick event handler of a link, but can also be referred to from an image map, set as the SUBMIT method of an embedded frame, or called from another user-defined JavaScript function.

Refer to the Referencing Other Siebel Data chapter in *Siebel Interactive Designer Administration Guide* for more information.

Syntax

GotoSSConfigurator(*productDescriptionString*)

Argument	Description
productDescriptionString	Optional argument used to describe the customizable product, which is used to start up the server-side configurator. If ProductDescriptionString is not used, the current model and state of the application are used to build a customizable product structure, which is then passed on to the server-side eConfigurator to launch the runtime instance of the customizable product. You can define ProductDescriptionString in the following ways: ■ As product ID ■ As a fully formed string, built by hand or using helper APIs (see "BuildProductStr" on page 51). Use the form: `relationship=<relationship name>&*prodID=<prodID>&*qty=<quantity>&*attributes={<key1>=<value1>&*<key2>=<value2>&*...}&*children={<child1>|*<child2>|*...}` ■ As null to specify not to extract a string

Example

```
ISS.GotoSSConfigurator();
ISS.GotoSSConfigurator(ISS.GetResultsValue(CONFIG_PROD))
ISS.GotoSSConfigurator(ISS.GetBusCompID(”ROOT_PART”));
```

GoToSSView

Usage Use the GoToSSView() function to switch from the current browser-based view to any specified Siebel application view. All Siebel views have an exact name. This does not select a particular record in that view.

Syntax GoToSSView(*ViewName*)

Argument	Description
ViewName	A string with the view name.

Example

```
ISS.GotoSSView("Product Detail View");
```

SendSelectionInformationToServer

Usage Use the SendSelectionInformationToServer function to post specified data to a business service and method.

NOTE: To use this function, you need to write a business service and a method on that service.

You can use this function to create a variety of integration points with other applications as follows:

- Create a new opportunity in Siebel Sales as a result of a web advisor session.
- Store results of a session to a Business Component other than Quotes.
- Call a string of Dynamic Defaults from a Business Component and open a pageset that displays them.
- Pass an optional field promoCode (promotional code) to the method so that it will specially price the information based on that code.
- Use a CDA Service Broker in the ISSCDA RT UI Service to:
  - Call another business service.
  - Accept information from a called business service.
  - Return a confirmation response to the client.

Syntax SendSelectionInformationToServer(*service, method, prodStr, enableLinkback, optArgs*)

Argument	Description
service	Name of the Business Service to be executed.
method	Method to be used on the business service.

Argument	Description
prodStr	Product string describing the product. Use null, without quotes around it, to specify that no product string should be sent. Use “auto” to automatically extract the product string. prodStr is added automatically when you choose to use it in the other arguments. Do not set it manually.
enableLinkback	Boolean value specifying whether to build a linkback string used for restoring state. If “true,” constructs and passes the appropriate linkback string. If “false,” takes no action. The linkback string gets added automatically when you choose to use it in the other arguments. It should not be set manually.
optArgs	Object including at least the following special fields: prodStr and linkBackStr. Any additional data should be included as fields in this object as a string or number. This data will be associated with the field name when sent to the method.

Example To send the model and year of your car along with product and state information, call the function as follows:

```
var optArgs = new Object();
optArgs["carModel"] = "Sedan";
optArgs["Year"] = 99;
ISS.SendSelectionInformationToServer("myService", "myMethod",
"auto", true, optArgs);
```

This call will automatically extract product information from the data model since auto is true. It will also create a linkback string for the current state of the pageset. The linkback string can then be used to open the pageset in this state.

Using the CDA Service Broker in the ISSCDA RT UI Service

You can call the CDAServiceBroker method in the ISSCDA RT UI Service to have the client receive a confirmation response after it posts a request to a business service using SendSelectionInformationToServer().

The CDA Service Broker in the ISSCDA RT UI business service acts as a broker to:

- Call another business service
- Accept information from that business service
- Return a confirmation response back to the client

The following examples illustrate how to accomplish these tasks using the CDA Service Broker.

To invoke the CDA Service Broker, you must set the following parameters in the SendSelectionInformationToServer function:

```
Service="ISSCDA RT UI Service"

Method="CDAServiceBroker"
```

Example 1 CDANoReturn

Use this parameter when no confirmation is needed from the server business component.

```
argObj["CDAExternalServiceName"] = "My Business Service";

argObj["CDAExternalMethodName"] = "My Business Service Method";

argObj[“CDANoReturn”]= “”;

OL.SendSelectionInformationToServer("ISSCDA RT UI Service",
"CDAServiceBroker", "auto", true, argObj);
```

Example 2 CDAReturnValue

This parameter requests the business service to select a value from the output property set and return it to the client.

```
argObj["CDAExternalServiceName"] = "My Business Service";

argObj["CDAExternalMethodName"] = "My Business Service Method";

argObj[“CDAReturnValue”]= “ReturnName”;

OL.SendSelectionInformationToServer("ISSCDA RT UI Service",
"CDAServiceBroker", "auto", true, argObj);
```

In this example, the client specifies CDAReturnValue = ReturnName. After successful execution of the method, the client can retrieve the returned property value from ReturnName.

Example 3 CDAReturnHandler

If you need to retrieve the entire output property set from the server, you can build a property set handler for the client side. The name of the property set handler is passed to the server side to process the property set object. The ISSCDA RT UI Service reconstructs the property set into JavaScript objects and sends it back to the client.

```
argObj["CDAExternalServiceName"] = "My Business Service";

argObj["CDAExternalMethodName"] = "My Business Service Method";

argObj[“CDAReturnHandler”]=
OL.FrameToString(window)+'.alertEvent');

OL.SendSelectionInformationToServer("ISSCDA RT UI Service",
"CDAServiceBroker", "auto", true, argObj);

function alertEvent(ps)

{

   if (ps != null)

   {

       for (bFirst = true; (arg = ps.EnumProperties (bFirst)) !=
null; bFirst = false)

       {

            value = ps.GetProperty (arg);

            alert(arg + "=" + value);

        }

    }

    else

    {
```

```
        alert("Empty PropertySet in function alertEvent");
      }
} </script>
```

NOTE: The JavaScript object constructed from the server side code uses the constructor (which resides in the propset.js) in the SWE script. Be sure to include this script in the file where the property set handler is in your JavaScript.

Example 4 Null value

If you use a null value for optArgs, the CDA Service Broker retrieves the default property set name from the repository. The default name is defined in the Siebel repository under the ReturnPropertyName in the user property of the business service CDA RT UI Service. The CDA Service Broker calls the specified business service using the default property set name. If the default property set name in the specified business service exists in the return output property set, it will be returned to the client.

```
argObj["CDAExternalServiceName"] = "My Business Service";

argObj["CDAExternalMethodName"] = "My Business Service Method";

OL.SendSelectionInformationToServer("ISSCDA RT UI Service",
"CDAServiceBroker", "auto", true);
```

ShowCDA

Usage Use the ShowCDA function to load a specified pageset using a linkback string.

Syntax ShowCDA(*pagesetName,linkbackString,args[]*)

Argument	Description
pagesetName	Name of the pageset, specified in the format "project|pageset" (for example, Cars|Sedans). If "project|pageset" is not specified, showCDA loads the personalized contents list of the specified project. If a NULL parameter is specified, ShowCDA loads the default project.
linkbackString	Optional argument. linkbackString is the string used to restore a previous state. Use the same string the browser-based engine passed off to the server and stored in the database.
args	Optional argument. Add a ShowContents property to the args[] object and set it to "true" in order to display a contents list when the pageset appears. By default, this variable is set to false. Use the GetCDAEntryArgs() and GetCDAEntryArg() functions to return the argument array.

Example

```
var optArgs = new Object;

optArgs.ShowContents = true;

ISS.ShowCDA("Cars|Sedans", null, optArgs);
```

ShowCDAWithDynDefStr

Usage Use the ShowCDAWithDynDefStr function to load a specified pageset using a Dynamic Default string.

Syntax ShowCDAWithDynDefStr(*pagesetName,dynDefStr,args[]*)

Argument	Description
pagesetName	Name of the pageset, specified in the format “project|pageset” (for example, Cars|Sedans). If “project|pageset” is not specified, showCDA loads the personalized contents list of the specified project. If a NULL parameter is specified, ShowCDA loads the default project.
dynDefStr	Optional argument. String that defines key-value pairs of input Feature table columns and their values. The default format of *dynDefStr* is "key1 = value1,key2 = value2". Separator characters default to equal signs (=) and commas (,).
args	Optional argument. Add a ShowContents property to the args[] object and set it to “true” in order to display a contents list when the pageset appears. By default, this variable is set to false. Use the GetCDAEntryArgs() and GetCDAEntryArg() functions to return the argument array.

Example

```
ISS.ShowCDAWithDynDefStr(“Cars|Sedans”);

ISS.ShowCDAWithDynDefStr(“Cars|Sedans”, “COLOR=BLUE, ENGINE=V6”);
```

ShowProductDetails

Usage

Use the ShowProductDetails function to open the detail view for the currently selected product ID. This API uses the parameters defined in the Siebel config file to determine which Siebel application view to open. The parameters are the same for all browser-based applications within the context of a specific Siebel application. For example, within Call Center, all browser-based applications executing this API would open the same Detail View. If the Detail View defined in uagent.cfg (the config file for Call Center) is the Product Detail View, then the API always submits a request to switch the view to the Product Details View.

ShowProductDetails can be called from any frame within the application and can be executed anywhere a JavaScript function can be used. It is commonly called from the onClick event handler of a link, but can also be referred to from an image map, set as the SUBMIT method of an embedded frame, or called from another user-defined JavaScript function. Unless you pass in a product string, the link to ShowProductDetails should be implemented in places that are active only if there is a valid product.

Syntax

ShowProductDetails(*product*)

Argument	Description
product Id	The product ID of the product. This is an optional argument. If no parameter is specified, the root product id is used.

Example

```
ISS.ShowProductDetails(ISS.GetBusCompID("MODEL.DESC"));

ISS.ShowProductDetails(ISS.GetResultsValue(”PROD_DETAIL_ID”));
```

ShowProductDetails

4 Utility Functions

Use utility functions at various points in a browser-based application to support other browser-based application functions.

ConvertFloatToCurrency

Usage Use the ConvertFloatToCurrency function to display numbers in output targets in currency format, to two decimal places. If you do not use the ConvertFloatToCurrency function, you must write a custom function to format and display output target numbers in currency format.

Use ConvertFloatToCurrency by itself or inside a BuildTarget function to format values before they appear in the application.

Syntax ConvertFloatToCurrency(*value*)

Argument	Description
value	Value to be converted, in floating-point form.

Example The following sample code uses ConvertFloatToCurrency with the BuildTarget function to convert and return numbers in currency format:

```
<SCRIPT language="JavaScript">

document.write(ISS.BuildTarget("TEXT",window,"PRICE",
ISS.ConvertFloatToCurrency));

</SCRIPT>
```

In this example, ConvertFloatToCurrency returns numbers in text output targets in currency format. For example:

- 31.1 would return 31.10
- 20 would return 20.00
- 20.50 would return 20.50
- 20.605 would return 20.61

See Also "BuildTarget" on page 106.

ConvertStrToDynDefObj

Usage Use the ConvertStrToDynDefObj function to convert a string of arguments into an object for use in the LoadPagesetWithDynDefObj function.

The LoadPageset function also translates strings to objects for you. Use the ConvertStrToDynDefObj function only when you are writing substantial amounts of custom code and need to convert a string of arguments into an object that you will subsequently modify.

Syntax ConvertStrToDynDefObj(*str*)

Argument	Description
str	String to be converted. Default format of *str* is "key1 = value1,key2 = value2." Separator characters default to equal signs and commas.

Example

```
ISS.ConvertStrToDynDefObj("PETTYPE=TABBY,SIZE=MEDIUM,
COLOR=ORANGE");
```

See Also "LoadPagesetWithDynDefObj" on page 131.

"LoadPageset" on page 129.

FrameToOLString

Usage Use the FrameToOLString function to determine the appropriate Document Object Model (DOM) path to a particular frame.

Browser-based applications establish a virtual document root at ISS. In many instances, you need to define a path relative to ISS. This function performs the necessary translation.

This function returns a string value.

Syntax

FrameToOLString(*frame*)

Argument	Description
frame	DOM-based window object, returns a string.

Example The Application UI Definition file (/ui/ol_ui.htm) contains the following reference:

```
ISS.RegisterUI(ISS.FrameToOLString(window)+'.mainArea')
```

In a standard browser-based application, this is equivalent to:

```
ISS.RegisterUI(ISSStr+'.ol_ui.mainArea');
```

GetCSPath

Usage Use the GetCSPath function to return the file system path to the cs directory of the application.

Use this function instead of hard-coding a relative directory path (for example, ../cs/dog.js), because the relative path can change depending on:

- The location from which the function is called
- The browser in which the application is running

Syntax GetCSPath()

Example If the application is installed at `http://server.company.com/myapp`,you could define

```
var cspath = ISS.GetCSPath();
```

In this case, *cspath* would be the string "/myapp/cs/".

GetCustomPath

Usage Use the GetCustomPath function to return the file system path to the custom directory of the application.

Use this function instead of hard-coding a relative directory path (for example, ../custom/app_config.js), because the relative path can change depending on:

- The location from which the function is called
- The browser in which the application is running

Syntax GetCustomPath()

Example If the application is installed at `http://server.company.com/myapp`, you could define

```
var custompath = ISS.GetCustomPath();
```

In this case, *custompath* would be the string "/myapp/custom/".

GetDSPath

Usage Use the GetDSPath function to return the file system path to the ds directory of the application.

Use this function instead of hard-coding a relative directory path (for example, ../ds/dog_x.js), because the relative path can change depending on:

- The location from which the function is called
- The browser in which the application is running

Syntax GetDSPath()

Example If the application is installed at `http://server.company.com/myapp`, you could define

```
var dspath = ISS.GetDSPath();
```

In this case, *dspath* would be the string "/myapp/ds".

GetJDPath

Usage Use the GetJDPath function to return the file system path to the jd directory of the application.

Use this function instead of hard-coding a relative directory path (for example, ../jd/header.js), because the relative path can change depending on:

- The location from which the function is called
- The browser in which the application is running

Syntax GetJDPath()

Example If the application is installed at `http://server.company.com/myapp`, you could define

```
var jdpath = ISS.GetJDPath();
```

In this case, *jdpath* would be the string "/myapp/jd".

GetPagesetDisplayArea()

Usage GetPagesetDisplayArea() returns the string representing the frame location where the pagesets should appear. It is set by RegisterUI. The frameset registered in the _i file loads in that frame.

Syntax GetPagesetDisplayArea()

GetPGPath

Usage Use the GetPGPath function to return the file system path to the pg directory of the application.

Use this function instead of hard-coding a relative directory path (for example, ../pg/dog_1.htm), because the relative path can change depending on:

- The location from which the function is called
- The browser in which the application is running

Syntax GetPGPath()

Example If the application is installed at `http://server.company.com/myapp`, you could define

```
var pgpath = ISS.GetPGPath();
```

In this case, *pgpath* would be the string "/myapp/pg".

GetTopPath

Usage Use the GetTopPath function to return the file system path to the top-level application directory.

Use this function instead of hard-coding a relative directory path (for example, ../home.htm), because the relative path can change depending on:

- The location from which the function is called
- The browser in which the application is running

Syntax GetTopPath()

Example If the application is installed at `http://server.company.com/myapp`, you could define

```
var toppath = ISS.GetTopPath();
```

In this case, *toppath* would be the string "/myapp/".

GetUIPath

Usage Use the GetUIPath function to return the file system path to the ui directory of the application.

Use this function instead of hard-coding a relative directory path (for example, ../ui/ol_ui.htm), because the relative path can change depending on:

- The location from which the function is called
- The browser in which the application is running

Syntax GetUIPath()

Example If the application is installed at `http://server.company.com/myapp`, you could define

```
var uipath = ISS.GetUIPath();
```

In this case, *uipath* would be the string "/myapp/ui".

GetVisibleDisplayArea()

Usage GetVisibleDisplayArea returns a string representing the frame that displays the entire UI.

Syntax GetVisibleDisplayArea()

RegisterUI

Usage Use the RegisterUI function to register the user interface frameset name in the Application UI Definition file (\ui\ol_ui.htm). The value set here can be later accessed using ISS.GetPagesetDisplayArea(). RegisterUI tells the engine that the user interface frameset has loaded completely and which frame will contain the display pages for pagesets.

Use RegisterUI in the onLoad handler of \ui\ol_ui.htm.

Syntax ISS.RegisterUI(*frameName*)

Argument	Description
frameName	Full path to the frame in which the pageset frameset for display pages will be loaded. Path must be expressed relative to ISS.

Example The frameset of \ui\ol_ui.htm might include the following code:

```
<FRAMESET cols=”240,*” frameborder=0 framespacing=0
onLoad="ISS.RegisterUI(ISS.FrameToOLString(window)
+'.mainArea');”>
```

See Also “FrameToOLString” on page 76.

RegisterUIElement

Usage

For refresh purposes, use the RegisterUIElement function to inform the application whether a user interface element is an input UI control or an output target.

Pages that do not contain output targets do not usually need to be refreshed after the engine executes. For performance and UI benefits, applications track which pages have only input UI controls, and therefore do not need to be refreshed.

If you write a custom input UI control or target, you must call RegisterUIElement to inform the application whether your custom creation needs to be refreshed.

Syntax

RegisterUIElement(*elementName,window,mustRefresh*)

Argument	Description
elementName	For input UI controls, the name of the Feature table from which the UI control is built. For output targets, the name of the Configuration table column, Feature table, or Feature table column for which the target is built.
window	The DOM window object that contains the input UI control or output target.
mustRefresh	Boolean variable used to specify whether *elementName* is an output target (true = output target, false = input UI control).

See Also

"Example 1: Create a Custom Input UI Control" on page 174

"Example 3: Create a Custom Output Target" on page 178

ShowAbout

Usage Use the ShowAbout function to open a separate window and display the information contained in the \ui\about.htm file. The About information commonly contains the application version number, legal notices, and company contact information.

You can call the ShowAbout function from any frame within an application, and the function can be executed anywhere a JavaScript function can be used. It is commonly called from the onClick event handler of a link, but it can also be referred to from an image map, set as the SUBMIT method of an embedded frame, or called from another user-defined JavaScript function.

Syntax ShowAbout()

Notes

- The information that appears when the ShowAbout function is called must be located in an HTML file called about.htm, and this file must be located in the application ui subdirectory.
- When you create a new project, Interactive Designer automatically generates an about.htm file in the application ui subdirectory. Edit this file to contain information appropriate for your own application.
- The configuration variable APP_ABOUT_WIN_ARGS in the Application Configuration file (\custom\app_config.js) defines the attributes of the About window, including size, whether a status bar and scroll bars appear, and whether the window is resizable.

Example The following sample code creates a link to the About window using a link surrounding a button image called btn_abt.gif.

```
<A HREF="javascript:''" onClick="ISS.ShowAbout( ); return
false;">
<IMG src="btn_abt.gif" alt="Information about CDA(tm)" height=14
width=57 border=0 hspace=0 vspace=0 align=top> </A>
```

ShowHelp

Usage Use the ShowHelp function to open a separate window displaying the information contained in the \ui\helpset.htm file. The helpset contains information about the application user interface and the controls that it contains, and you can edit the generic information to customize the help for your own application.

You can call the ShowHelp function from any frame within the application, and the function can be executed anywhere a JavaScript function can be used. It is commonly called from the onClick event handler of a link, but it can also be referred to from an image map, set as the SUBMIT method of an embedded frame, or called from another user-defined JavaScript function.

Syntax ShowHelp()

Notes

- The information that appears when the ShowHelp function is called must be located in the file helpset.htm, and this file must be located in the application ui subdirectory.
- The configuration variable APP_HELP_WIN_ARGS in the Application Configuration file (\custom\app_config.js) defines the attributes of the Help window, including size, whether a status bar, menu bar, and scroll bars appear, and whether the window is resizable.
- The helpset.htm file generated when you create a new project in Interactive Designer contains a frameset that refers to the following source files: help_top.htm, help_mnu.htm, help_ovr.htm, and help_nav.htm. Using a frameset is optional. All of the Help information can appear in the helpset.htm file only.

Example The following sample code creates a link to the Help window using a link surrounding a button image called btn_hlp.gif.

```
<A HREF="javascript:'" onClick="ISS.ShowHelp( ); return
false;"><IMG src="btn_hlp.gif" alt="browser-based application
Help" height=14 width=57 border=0 hspace=0 vspace=0 align=top></
A>
```

ShowHelp

User Interface Layout and Control Functions 5

User interface layout and control functions define the general structure and appearance of an application. If you want to modify the default layout, you can use these functions to define virtually any interface.

RegisterCascade

Usage Use the RegisterCascade function to identify the name of a Cascade Registry Table. Required for any pageset that has a cascading trigger construct. RegisterCascade is called from the UI Information file for the pageset. Multiple calls can be made to register multiple cascading trigger constructs.

Syntax ISS.RegisterCascade(*cascadeName*);

Argument	Description
cascadeName	The name of the cascade trigger table.

RegisterExceptionFrames

Usage

Use the RegisterExceptionFrames function to define which frame in a browser-based application displays exception messages. This function is always called from the Pageset UI Registry file (\pg*pagesetID*_i.htm).

A frame is often both a normal display frame and also an exception frame. For example, a pageset user interface might contain two frames: one frame containing only input UI controls and another containing only output targets. If you register the output target frame as an exception frame, it displays results such as prices and pictures for a valid configuration but is completely overwritten to display exception messages for an invalid configuration.

You cannot use a single-frame user interface for a pageset that can produce exception messages.

Syntax

RegisterExceptionFrames(*frameName*)

Argument	Description
frameName	The name of the relevant frame. This is a string. It is not a pointer. This should be a full path using ISS.GetPagesetDisplayArea.

Example

The following sample code uses RegisterExceptionFrames to indicate that exception data should be written into the uidata nested frame of the mainArea frame.

```
ISS.RegisterExceptionFrames
(ISS.GetPagesetDisplayArea()+".uidata");
```

See Also

"Example 2: Define a Pageset Layout" on page 176

RegisterFrameSet

Usage Use the RegisterFrameSet function to identify the file that contains the pageset UI definition frameset. This function is always called from the Pageset UI Registry file (\pg*pagesetID*_i.htm). Each Pageset UI Registry file must contain a RegisterFrameSet function call.

Each pageset must point to a Pageset UI Definition file (located in the pg directory) that defines how to subdivide the pageset display area defined in the Application UI Definition file (\ui\ol_ui.htm). More than one pageset can point to the same Pageset UI Definition file, or each pageset can have its own UI definition. RegisterFrameSet identifies the HTML file that is loaded into this area. This HTML file must always contain a frameset definition that indicates how to subdivide the space.

Syntax RegisterFrameSet(*name*,*frameSetName*,*frameLoc*,[*path*])

Argument	Description
name	An identifier. When the pageset UI definition frameset finishes loading, it calls a registration function that passes *name* as one of its arguments. Usually, this identifier name is the same as the name of the HTML file that contains the frameset.
frameSetName	Name of the HTML file containing the frameset.
frameLoc	A string. This argument is the fully qualified name of the target frame into which the frameset will be loaded. In standard applications, this refers to the mainArea frame defined in the Application UI Definition file (\ui\ol_ui.htm).
path	Optional. This argument is the full path to the directory in which the HTML file resides (pg, by default).

Example The following sample code uses RegisterFrameSet to load the dogs_ui.htm Pageset UI Definition file into the mainArea frame of an application.

```
ISS.RegisterFrameSet("dogs_ui","dogs_ui.htm",
ISS.GetPagesetDisplayArea());
```

The following example points to a frameset file that is located in the ui directory.

```
ISS.RegisterFrameSet("dogs_ui","dogs_ui.htm",
ISS.GetPagesetDisplayArea(),ISS.GetUIPath());
```

See Also "RegisterContentsListFrame" on page 134

"Example 2: Define a Pageset Layout" on page 176

RegisterMVar

Usage Use the RegisterMVar function to identify the name of a Multi-Variable Trigger Table. Required for any pageset that has a multi-variable trigger construct. RegisterMVar is called from the UI Information file for the pageset. Multiple calls can be made to register multiple multi-variable trigger constructs.

Syntax ISS.RegisterMVar(*mVarName*);

Argument	Description
mVarName	The name of the multi-variable trigger table.

RegisterPageLocation

Usage Use the RegisterPageLocation function to map specific display pages into the frames defined for a pageset. This function is always called from the Pageset UI Registry file (\pg*pagesetID*_i.htm). This function also defines which pages are visible when the pageset first loads.

You must call the RegisterPageLocation function once for each display page in the pageset, even if a page is not initially visible. You can rotate two or more display pages through a single pageset frame.

Syntax RegisterPageLocation(*fileName,frameName,isVisible,[path]*);

Argument	Description
fileName	Name of the HTML display page.
frameName	Name of the frame in which the HTML display page appears.
isVisible	String that indicates whether or not the file is visible when the pageset first loads ("true" = visible, "false" = invisible).
Path	Optional. The full path to the directory (pg by default) containing HTML file.

Example The following sample code registers the dogs.htm display page (located in the pg directory) into the ui_disp nested frame of the main display page frame (mainArea).

```
ISS.RegisterPageLocation("dogs.htm",
ISS.GetPagesetDisplayArea()+".uidisp","true");
```

The following example registers a display page that is located in the ui directory instead.

```
ISS.RegisterPageLocation("dogs.htm",
ISS.GetPagesetDisplayArea()+".ui_disp","true",ISS.GetUIPath());
```

See Also "Example 2: Define a Pageset Layout" on page 176

RegisterPriorityPages

Usage Use the RegisterPriorityPages function to identify the display pages that should be loaded before engine results are calculated. This function is always called from the Pageset UI Registry file (\pg*pagesetID*_i.htm).

The RegisterPriorityPages function is most frequently used to indicate which pages should be loaded first. The function often points to display pages containing only input UI controls whose content does not change based on engine results.

Syntax RegisterPriorityPages(*fileNames*)

Argument	Description
fileNames	A single string containing one or more comma-separated file names.

Example The following sample code uses RegisterPriorityPages to make sure that cats.htm and dogs.htm load before other display files.

```
ISS.RegisterPriorityPages("cats.htm,dogs.htm");
```

See Also "Example 2: Define a Pageset Layout" on page 176

Pageset Functions 6

This chapter discusses pageset functions, which allow for a flexible user interface in a browser-based application. Some pageset functions load and unload pagesets, while others map feature and configuration data to user interface elements in a browser-based application.

Start On Active

Several functions comprising the Start On Active function allow you to launch your application at a particular pageset, from an external URL, with or without a preconfigured setting.

For those instances where no pageset ID is specified, you can pass a default start page as a parameter in the SOALoadPageset function. Start On Active supports a variety of parameter passing formats. In addition, you can implement custom URL string parsing functionality using Override Points.

Start On Active can be called from any frame within the application and can be executed anywhere a JavaScript function can be used. It is commonly called from the onClick event handler of a link, but it can also be referred to from an image map, set as the SUBMIT method of an embedded frame, or called from another user-defined JavaScript function.

Start On Active is used to load a particular pageset. The pageset ID is explicitly stated in the function call. To load a pageset with an ID that is determined by the valid configuration of another product, create a link output target using the BuildTarget function.

Parameter Passing Formats

The following URL parameter passing formats are supported, where home.htm is the top level frameset:

http://…/home.htm

Launches the default application.

http://…/home.htm?pagesetid = xxx

Launches the application at pageset xxx with default Feature table selections as defined on the pageset.

If information not related to the Start on Active function is being passed in the URL query string, it should precede the pagesetid key/value pair (for example, a session ID).

http://…/home.htm?pagesetid = xxx&param1 = yyy&param2 = zzz

Launches the application at pageset xxx with preconfigured settings.

The syntax param1 = yyy corresponds to FeatureTable/CodeValue pairs. Code value yyy is selected for Feature table param1. Code value zzz is selected for Feature table param2. If information not related to the Start on Active function is being passed in the URL query string, it should precede the pagesetid key/value. Similarly, pagesetid should precede all dynamic default key/value pairs.

http://…/home.htm?pagesetid = xxx&dyndefs = param1 + yyy + param2 + zzz

Launches the application at pageset xxx with a preconfigured object.

The syntax param1 + yyy corresponds to FeatureTable/CodeValue pairs in the preconfigured object.

Information not related to the Start on Active function may appear anywhere in the query string.

http://…/
home.htm?pagesetid = xxx&configvars = true&param1 = yyy&param2 = zzz

Launches the application at pageset xxx with preconfigured settings.

The syntax param1 = yyy corresponds to FeatureTable/CodeValue pairs. The key/value pair configvars = true in the query string is used to indicate that a preconfigured setting is appended to the URL. The preconfigured setting key/value pairs must immediately follow the configvars = true parameter. If the parameter configvars is set to false, the application will launch at pageset xxx with the default settings as defined on the pageset. If information not related to the Start on Active API is being passed in the URL query string, it should precede the configvars key/value pair.

http://…/home.htm?config_file_path = filename.htm

Loads filename.htm, then launches the application at the pageset defined in that document's form, with the preconfigured settings also defined in that form.

The path for filename.htm should be fully qualified. This approach is particularly valuable in circumventing the URL length restrictions of most browsers (generally 2,000 characters in Microsoft Internet Explorer and Netscape Navigator).

Implementing Start On Active

To implement the Start On Active function, use the following methods.

Modify InitApp()

Modify the function InitApp() in custom/customCode.js to call ISS.SOALoadPageset(). The function takes two optional parameters:

- The default start page if no pageset is specified in the query string.
- A Boolean value indicating whether this default page is an active pageset (as opposed to a Splash screen).

An example of how this function may look is shown below:

```
function InitApp() {

   // insert initialization code here

   // start with contents listing

   ISS.ShowContentsList();

   // can start with page in addition to or instead of showing
   contents list

   ISS.SOALoadPageset(ISS.GetUIPath()+"welcome.htm", false);

}
```

To avoid a flash effect, verify that the source of the mainArea frame in ol_ui is "`javascript:`"

Use Hidden Frames

To implement Start on Active with hidden frames, create a new hidden frame in ol_ui called "dynaframe" using the parameter string format http://…/ home.htm?config_file_path = filename.htm.

NOTE: Be sure to add space for the hidden frame in the frameset's row/column definition.

The source for this file should be the empty javascript call:

```
<FRAME SRC="javascript:'" NAME="dynaframe" MARGINWIDTH="0"…>
```

Start on Active File Format Conventions

The contents of any Start on Active file specified in the config_file_path parameter of the URL query string should adhere to the following convention:

```
<SCRIPT src="jd/header.js"></SCRIPT>
```

NOTE: Be sure to include the header.js file and verify that the path is correct relative to the location of your Start on Active file.

Call "ISS.SOAPassDynaObject()" in the file's BODY onLoad handler:

```
<BODY onLoad="ISS.SOAPassDynaObject()">
```

Specify all key/value pairs as hidden INPUT fields. If information not related to the Start on Active function is included in the form, it should precede the pagesetid key/value. Similarly, pagesetid should precede all dynamic default key/value pairs.

The following is an example of a Start on Active file:

```
<HTML>
   <HEAD>
      <SCRIPT></SCRIPT> <SCRIPT src="jd/header.js"></SCRIPT>
   </HEAD>
   <BODY onLoad="ISS.SOAPassDynaObject()">
   <FORM name=configuration>
      <INPUT type=hidden name=pagesetid value=soa2>
      <INPUT type=hidden name=A value=3>
      <INPUT type=hidden name=B value=2>
      <INPUT type=hidden name=C value=1>
   </FORM>
   </BODY>
</HTML>
```

Start on Active File Function Overview

InitApp(), a user-defined function customCode.js in the custom directory, is used to define the start state of the application. The API ISS.SOALoadPageset() calls Start on Active from the InitApp() function.

SOALoadPageset calls GetParmStr() to retrieve the URL query string. Before the string is returned, it is cleaned of special characters using the API CleanQuery(). If there is no query string, the default start page is passed as a parameter in ISS.SOALoadPageset() and loaded into the ISS.uiFrame. Otherwise, StartActive() is called to evaluate the query string.

StartActive first checks whether the query string matches the parameter pattern for the hidden frame approach. If so, the function LoadSOAFrame() is called to load the supplied file into ol_ui.dynaframe. This approach relies on the onLoad handler of the supplied file to call SOAPassDynaObject(). SOAPassDynaObject will parse the form contents into query string format and pass that string back into StartActive for evaluation.

If StartActive does not match the config_file_path pattern in the query string, it calls the GetPagesetID() function to parse out the pagesetid key/value pair. If the key/value pair is found, GetConfigObj() function is called to evaluate the string for dynamic defaults. If the pagesetid key/value pair cannot be retrieved, the default start page is loaded.

GetConfigObj() evaluates the query string against the supported parameter formats. If a pattern match is found, an ISS.DynDef_Obj is constructed with the key/value pairs specified in the query string. This object is returned to StartActive.

Finally, StartActive checks whether a configuration object was returned. If so, it calls ISS.LoadPagesetWithDynDefObj and passes the pagesetid and configuration object as parameters. If not, it calls ISS.LoadPageset with the pagesetid.

NOTE: All calls to the Start On Active function should be of the form `ISS.FunctionName(arg1,…argN);`

SOALoadPageset (defaultPg, isActive)

Usage Use this function to launch the Start On Active pageset functionality.

This function is called from the InitApp() function in customCode.js.

Syntax SOALoadPageset (*defaultPg, isActive*)

Argument	Description
default	The name of a default start page to occupy the main area of the application. If the page is a splash screen without ISS calls, it should be fully qualified filename with the path and extension (for example, ISS.GetUIPath() + 'welcome.htm'). If the page is an active pageset, specify the pageset name as you would for ISS.LoadPageset (for example, 'advisor'). This parameter is optional.
isActive	Boolean value indicating whether defaultPg is an Active pageset or a splash screen. This parameter is optional - defaults to false.

SOAPassDynaObject ()

Usage This function is called in the onLoad handler of a Start On Active file used to supply the parameters in the hidden frame methodology. It parses the contents of the document form into URL query string format and then passes the string on for StartActive evaluation.

Syntax SOAPassDynaObject ()

BuildTarget

Usage Use the BuildTarget function on a display page to create output target controls that display links, images, or text in response to a user selection.

You call the BuildTarget function in the < BODY > section of a display page. Also, all BuildTarget calls must be inside the < FORM > section of the HTML source of display pages. The function is usually called from within a JavaScript document.write method.

Syntax BuildTarget(*type,window,name[,arg1,arg2]*)

Argument	Description
type	Specifies the type of output target control to create. Valid control types are: ■ **LINK**: see "Link Output Targets" on page 108. ■ SUBCONFIG_LINK: see "Subconfiguration Link Output Targets" on page 110. ■ OPT_SUBCONFIG_LINK: see "Optional Subconfiguration Link Output Targets" on page 112. ■ PICT: see "Image Output Targets" on page 113. ■ TEXT: see "Text Output Targets" on page 114. ■ **NB_LINK**: see "N-Back Link Target" on page 117. ■ **NB_RETURN:** see "N-Back Return Target" on page 118.
name	Defines the source of the content that populates the output target, which can be any of the following: ■ A column in a Configuration table. In this case, the argument is simply the name of the column. ■ A column in a Feature table. In this case, the argument is the name of the Feature table and the name of the specific column inside of it, separated by a period, as in COLOR.DESC. ■ The Feature table used to populate an input UI control from which the user has made a selection. In this case, the argument is simply the name of the Feature table. ■ The name of a text entry input UI control in which text is typed and stored. In this case, the argument is the name you used to define the text entry input UI control. For syntax, see "Content Sources for Output Targets" on page 115.
arg1, arg2	Some output target types take additional arguments.

Link Output Targets

Set the *type* argument for the BuildTarget function to LINK to create a link output target. The table column specified by the *name* argument can contain pageset IDs or URLs. The *isPage* argument defines how the data is handled. The *dynDefCol* argument points to a column that contains information that dynamically overrides the defaults of the specified pageset.

The syntax used to create a link output target is:

BuildTarget("*LINK*",*window,name,isPage[,dynDefCol]*)

Argument	Description
name	Column in a Configuration table or Feature table that contains pageset IDs or URLs.
isPage	This optional argument is a Boolean value that determines whether data in the column identified by the *name* argument is a pageset (TRUE) or a URL (FALSE). If the data is a URL and you set the argument to FALSE, the URL loads into a separate window, instead of inside the main application, when you click the link output target.
dynDefCol	This optional argument is a string that specifies the name of a table column (or columns, separated by commas) in which a preconfigured condition is defined for the pageset. When you click a link target containing a preconfigured condition, the application overrides the default settings of input UI controls in the pageset being loaded with the settings defined in the preconfigured condition. The definition of a preconfigured condition must contain the name of the overriding Feature table and the feature (CODE) value of the desired override setting. If you define a preconfigured condition column in a Configuration table, the *dynDefCol* argument references only the name of the Configuration table column whose values override the defaults. If you define a preconfigured column in a Feature table, the *dynDefCol* argument must contain the name of the Feature table *and* the name of the specific column inside of it whose values override the defaults, separated by a dot (as in COLOR.PRECONFIG).

NOTE: Pagesets must be located in the application pg subdirectory.

Example The following sample code creates a link output target based on a Configuration table column named CLINK:

```
<SCRIPT>

document.write(ISS.BuildTarget("LINK",window,"CLINK",
true));

</SCRIPT>If you liked this thing, you'll definitely like these
other things.</a>
```

When the user selections match a valid configuration, a pageset ID in the CLINK column is returned. When the user clicks the button, the pageset is loaded and displayed in the application.

Subconfiguration Link Output Targets

Set the *type* argument for the BuildTarget function to SUBCONFIG_LINK to create a subconfiguration link output target. Use a subconfiguration link output target to navigate to the user interface of a child pageset from its parent pageset, or from the child pageset back to the user interface of its parent pageset.

The syntax used to create a subconfiguration link output target is:

`BuildTarget(`*`"SUBCONFIG_LINK",window,instanceName`*`)`

Argument	Description
instanceName	Name of the relative instance that describes the subconfigured child pageset.

NOTE: Pagesets must be located in the application pg subdirectory.

Example The following sample code links from a parent page to a BURGER child page:

```
<SCRIPT>

document.write(ISS.BuildTarget("SUBCONFIG_LINK",
window,"BURGER"));

</SCRIPT>Link to Hamburger</a>
```

Similarly, a child page links to its parent page using the following sample code:

```
<SCRIPT>

document.write(ISS.BuildTarget("SUBCONFIG_LINK",
window,"PARENT"));

</SCRIPT>Link to Burger Meal</a>
```

Subconfiguration link output targets can link a child pageset, or a nested parent pageset, to the topmost parent pageset by using the reserved word TOP as the *instanceName* argument:

```
Done selecting cheese for your burger?

<SCRIPT>

document.write(ISS.BuildTarget("SUBCONFIG_LINK", window,"TOP");)

</SCRIPT>Link to Burger Meal</a>
```

Also, subconfiguration link output targets can link one child pageset to another by using the syntax PARENT:CHILD as the *instanceName* argument:

```
Done creating your burger?

<SCRIPT>

document.write(ISS.BuildTarget("SUBCONFIG_LINK",
window,"PARENT:FRIES");)

</SCRIPT>Would you like fries with that?</a>
```

Optional Subconfiguration Link Output Targets

Set the *type* argument for the BuildTarget function to OPT_SUBCONFIG_LINK to create an optional subconfiguration link output target. Use an optional subconfiguration link output target when a pageset is used for both a subconfigured item and a standalone item.

For example, if your application allows users to build fast food value meals, suppose that a user wanted to buy a drink without purchasing any food. By using an optional subconfiguration target link to the parent pageset on the DRINK pageset, the same DRINK pageset could be used, regardless of whether the person is ordering a value meal.

A subconfiguration link back to the parent is not appropriate unless the user arrives at the pageset from a parent pageset. The optional link does not appear unless the user arrives from a parent pageset.

The syntax to create an optional subconfiguration link output target is:

```
BuildTarget("OPT_SUBCONFIG_LINK",window,instanceName,preText,postText)
```

Argument	Description
preText	HTML and text to be written before the optional subconfiguration link output target.
postText	HTML and text to be written after the optional subconfiguration link output target.

NOTE: Pagesets must be located in the application pg subdirectory.

Example Use the following sample code:

```
<SCRIPT>

document.write(ISS.BuildTarget("OPT_SUBCONFIG_LINK"
window, "PARENT","<TR><TD>","Link to Burger Meal</A>
</TD></TR>"));

</SCRIPT>
```

returns:

```
"<TR><TD><a HREF="javascript:InstanceWrapper
('PARENT','window')">Link to Burger Meal</A></TD>
</TR>"
```

if there is a parent. If there is not a parent, it returns an empty string (" ").

Image Output Targets

When the value for the *type* argument for the BuildTarget function is PICT, an image output target is created. The image output target also uses optional *Width* and *Height* arguments.

The syntax used to create an image output target is:

`BuildTarget(`*`"PICT",window,name[,width,height]`*`)`

Argument	Description
name	Column in a Configuration table or Feature table that contains the name of an image file.
width	Optional. Specifies the width of the image, in pixels.
height	Optional. Specifies the height of the image, in pixels.

Argument Notes

- If values for the *width* and *height* arguments are not defined, the application defaults to displaying the image at the actual size specified by the image file. In some browsers, rendering speed can be improved by explicitly specifying dimensions with the *Width* and *Height* arguments.
- The path to an image file is relative to the pg subdirectory.

Example The following sample code creates an image output target that displays the file listed in the IMG column of a Configuration table:

```
<SCRIPT>

document.write(ISS.BuildTarget("PICT", window,"IMG"));

</SCRIPT>
```

When the user selections match a valid configuration, the value in the IMG column is returned. This value is the name of the image file that appears in the output target.

Text Output Targets

When the *type* argument for the BuildTarget function is set to TEXT, a text output target control is created.

The syntax used to create a text output target is:

`BuildTarget(`*`"TEXT",window,name,[formatFunction]`*`)`

Argument	Description
name	Column in a Configuration table, column in a Feature table, or a text UI control.
formatFunction	Optional. A pointer to a function used to format the display of the value that is often used to format numeric values.

Example The following sample code creates a text output target that displays an interior color selection. When the user selections match a valid configuration, a value in the DESC column is returned and displayed in the text output target. Note that this column is located in the Feature table COLORINT.

```
<SCRIPT>

document.write(ISS.BuildTarget("TEXT", window,"COLORINT.DESC"));

</SCRIPT>
```

The following sample code displays a price text as a number with two decimal places.

```
<SCRIPT>

document.write(ISS.BuildTarget("TEXT", window,"PRICE",
ISS.ConvertFloatToCurrency));

</SCRIPT>
```

Content Sources for Output Targets

The *name* argument for the BuildTarget function refers to one of the following:

- A column in a Configuration table. In this case, the argument is simply the name of the column.

  If the output target control is populated with data from a column in a Configuration table, the *name* argument is the column name only. The structure of the reference is:

  ```
  ColumnName
  ```

  For example, if an image output target is based on a Configuration table column called IMG, the syntax might be:

  ```
  BuildTarget("PICT",window,"IMG")
  ```

- A column in a Feature table. In this case, the argument is the name of the Feature table and the name of the specific column inside of it, separated by a period.

  The reference structure is:

  ```
  TableName.ColumnName
  ```

  For example, if a text output target is based on a DESC column in the Feature table called COLOR, the syntax might be:

  ```
  BuildTarget("TEXT",window,"COLOR.DESC")
  ```

- The Feature table used to populate an input UI control from which the user has made a selection. In this case, the argument is simply the name of the Feature table.

  In this case, the data from the CODE column for that table is returned and displayed in the output target. The reference structure is:

  TableName

  For example, if a text output target is based on a CODE column in the Feature table called PROCESSOR, the syntax might be:

  ```
  BuildTarget("TEXT",window,"PROCESSOR")
  ```

- The name of a text entry input UI control in which text is typed and stored. In this case, the argument is the name you used to define the text entry input UI control.

  For example, the following sample code might be used to create a text box UI control:

  ```
  <SCRIPT>document.write(ISS.BuildWidget
  ("GETTEXT",window,"PLATETEXT",8))</SCRIPT>
  ```

  The text that a user enters into the UI control is then saved as part of the selection set with the name reference PLATETEXT. The following sample code creates a text output target that displays this data:

  ```
  <SCRIPT>document.write(ISS.BuildTarget
  ("TEXT",window,"PLATETEXT"))</SCRIPT>
  ```

N-Back Link Target

The N-Back Link target saves the input state of the current pageset and then load a specified pageset. You can create one or more N-Back Link targets per display page. The N-Back Link Target is used in conjunction with the N-Back Return target. For more information, see "N-Back Return Target" on page 118.

The syntax used to create an N-Back Link output target is:

```
BuildTarget (“NB_LINK”, winObj, linktext, pagesetcol, returntext)
```

Argument	Description
NB_LINK	Specifies that this function will be used to navigate to a new pageset.
winObj	The window from which the call is made.
linktext	The text to appear for the Link or Return HREF. If returntext is specified when a Link is built, linktext is ignored during the construction of the corresponding Return target.
pagesetcol	Specifies the field of the data model that holds the name of the pageset to link to.
returntext	Optional Specifies the text to display for the return HREF.

Example

```
document.write(ISS.BuildTarget("NB_LINK",window,
"Accessories","CROSS_SELL","DYNDEFS", ”Return to Server
Configuration”));
```

N-Back Return Target

The N-Back Return target is used to navigate back to the previous pageset and restore the saved input state for that page. You can create one N-Back Return target per display page.

An N-Back Return target will only be constructed if called from a pageset of which the "TOP" instance was traversed to by an N-Back Link. Otherwise an empty string is returned. For more information, see "N-Back Link Target" on page 117.

The syntax used to create an N-Back Return output target is:

```
BuildTarget (“NB_RETURN”, winObj, linktext, tokencol)
```

Argument	Description
NB_RETURN	Specifies that this function will be used to traverse back to the previous pageset.
winObj	The window from which the call is made.
linktext	The text to appear for the Link or Return HREF. If returntext is specified when a Link is built, linktext is ignored during the construction of the corresponding Return target.
tokencol	Optional Specifies the field of the data model that holds the key/value pairs of a preconfigured object for the link.

Example

```
document.write(ISS.BuildTarget("NB_RETURN",window,"Return"));
```

BuildWidget

Usage

Use the BuildWidget function to add the following input UI controls to a display page in a browser-based application:

- Image map
- List box
- Radio button
- Check box
- Text box

The image map, list box, radio button, and check box input UI controls allow a user to make feature selections in a browser-based application. The text box input UI control allows the user to enter data that can be reused in the application.

You call the BuildWidget function in the <BODY> section of a display page. Also, all BuildWidget calls must be inside the <FORM> section of the HTML source of display pages. The function is usually called from within a JavaScript document.write method.

Syntax

BuildWidget(*type,window,name[,arg1,arg2,arg3,arg4,arg5]*)

Argument	Description
type	Specifies the type of input UI control to create. Valid control types are: CHECKBOX: see "Check Box Input UI Controls" on page 120. MAP: see "Image Map Input UI Controls" on page 121. LISTBOX: see "List Box Input UI Controls" on page 122. RADIO: see "Radio Button Input UI Controls" on page 124. GETTEXT: see "Text Entry Input UI Controls" on page 126.
name	Defines the source of the content that populates the input UI control.
arg1 - 5	Some input UI control types take additional arguments.

Check Box Input UI Controls

The syntax used to create a check box input UI control is:

```
ISS.BuildWidget("CHECKBOX",window,name)
```

Argument	Description
name	Defines the source of the content that populates the input UI control. For check box input UI controls, this is the name of a Feature table in the Interactive Designer project.

Notes The Feature table associated with a check box input UI control defines whether the input UI control is checked or cleared. Therefore:

- Only the first two rows of the Feature table are significant for this input UI control.
- The first row represents the cleared state, and the value in its CODE column must be equivalent to false (F, FALSE, or 0).
- The second row represents the checked state, and the value in its CODE column must be equivalent to true (T, TRUE, or 1).
- When a value is specified in the DEFAULT column of one row, the state represented by the row is the state of the check box when the pageset is first loaded.

Example The following sample code creates a check box input UI control:

```
<TR>

<TD><FONT face=verdana size=2><B>Personalized Plates
</B><IMG src=trans.gif width=10></TD>

<TD><SCRIPT>  document.write(ISS.BuildWidget
("CHECKBOX",window,"PLATE"));</SCRIPT></TD>

</TR>
```

Image Map Input UI Controls

An image map input UI control functions in much the same way as a group of radio button input UI controls function. For example, an image map input UI control can take the form of a three-color item. Users would click on a color, instead of a radio button, to make a selection.

In the relevant Feature table, you must specify the UI control name and its map name, input shape, and coordinates. You can swap your own image files to represent a selected palette, but you must provide your own browser-specific code to use DHTML to swap images.

The syntax used to create an image map input UI control is:

```
ISS.BuildWidget("MAP",window,name,mapName,fileCol,
shapeCol,coordCol)
```

Argument	Description
name	Feature table name and the name of the input UI control that corresponds to the information represented by the image map (for example, COLOR).
mapName	Name of the image map. All image maps must have unique names. Image maps cannot be named map.
file	File name of the image (or name of a table column that contains the file name of the image) used.
shapeCol	Interactive Designer data column that contains the shape of the image map area.
coordCol	Interactive Designer data column that contains coordinates of the image map area.

Example The following sample code creates an image map input UI control whose image source file name is defined in the PICT column of the COLOR Feature table:

```
<SCRIPT>

document.write(ISS.BuildWidget("MAP",window,"COLOR",
"color_map","PICT","SHAPE","COORD"));

</SCRIPT>
```

The following sample code creates an image map input UI control that directly calls the image source file called Flag.gif. This file should be located the pg directory.

```
<SCRIPT>

document.write(ISS.BuildWidget("MAP",window,"COLOR",
"color_map","Flag.gif","SHAPE","COORD"));

</SCRIPT>
```

List Box Input UI Controls

The *name* argument is the name of the Feature table that contains the data used to populate the list. The data in the DESC column of this table is always used to populate the list. Additionally, you must specify *height*, *width*, and *prefill* arguments for the input UI control.

The syntax used to create a list box input UI control is:

```
BuildWidget("LISTBOX",window,name,height,width,prefill)
```

Argument	Description
name	Defines the source of the content that populates the input UI control. For list box input UI controls, this is the name of a Feature table in the Interactive Designer project. Items in the list box will be filled in sequentially using the entries in the DESC column of this table.
height	For list box input UI controls, this argument is an integer that specifies the number of visible rows for the list box. Set this argument to 1 to create a drop-down list.
width	Integer that specifies the width of the list box, as defined by the number of characters visible in the default browser font.
prefill	Boolean value that sets the width of the list box control to accommodate the widest item in the list. If *prefill* is TRUE and *width* is greater than the number of characters in the longest line of text appearing in the list box, the list box appears with a width equivalent to *width*. If *prefill* is FALSE, the width is set to accommodate the longest line of text that appears in the list box.

Notes

- In Internet Explorer 4.*x* and 5.*x* browsers, selections in the list box appear in the standard system font. In Netscape 4.*x*, selections appear in the font in which text at that location would normally appear—either in the font specified for the entire document, or a font specified by an enclosing < FONT > tag set.
- The DEFAULT column in the associated Feature table determines which item in the list box is selected when the pageset is initially loaded. If no row in the table contains the value DEFAULT, the item that corresponds to the first row in the table is selected.

Example

The following sample code creates a list box that draws feature data from the Feature table COLORINT. Note that the actual caption text is created using a separate HTML entry.

```
Choose an interior color:

<SCRIPT>

document.write(ISS.BuildWidget("LISTBOX",window,
"COLORINT",1,38,true));

</SCRIPT>
```

Radio Button Input UI Controls

You can create a radio button input UI control that uses a graphic caption for the radio button instead of a text caption.The graphic specified for the *imageFile* argument will be used for the button caption.

The syntax used to create a radio button input UI control with a text caption is:

```
ISS.BuildWidget("RADIO",window,name,indexRow[,imageFile])
```

Argument	Description
name	Defines the source of the content that populates the input UI control. For list box input UI controls, this is the name of a Feature table in the Interactive Designer project. A radio button input UI control is associated with a particular row in the table. The row is specified by the *indexRow* argument.
indexRow	The row in the specified Feature table for which the button is created. (Note that the first row in the table is considered row '0'.)
imageFile	Optional. Interactive Designer data column, such as PICT, that contains information about the graphic, such as the name of the image file.

Notes

- A separate BuildWidget function call is required for each row in a Feature table that is associated with a radio button.
- The DEFAULT column in the associated Feature table determines which radio button is selected when the pageset is initially loaded. If no row in the table contains the value DEFAULT, the radio button that corresponds to the first row in the table is selected.

Examples The following sample code creates a radio button input UI control. This input UI control is based on the first row of the Feature table BODY, and the *indexRow* argument is set to *0*. The caption text is created using a separate HTML entry.

```
<SCRIPT>

document.write(ISS.BuildWidget("RADIO",window,"BODY",0));

</SCRIPT>2-Door Coupe

The following sample code creates the same 2-Door Coupe radio
button as in the previous example, but with a graphic caption.

<SCRIPT>

document.write(ISS.BuildWidget("RADIO",window,
"BODY",0,"PICT"));

</SCRIPT>

The PICT argument will first attempt to get the graphic from
BODY.PICT. If there is no PICT column in BODY, it will get the
graphic directly.

<SCRIPT>

document.write(ISS.BuildWidget("RADIO",window,"BODY",0,
"TwoDoorCoupe.gif"));

</SCRIPT>
```

TwoDoorCoupe.gif should reside in the pg directory.

Text Entry Input UI Controls

A text entry input UI control allows you to enter text in the application. This text is stored as a value in the selection set under the name specified as the *name* argument.

The syntax used to create a text entry input UI control is:

```
BuildWidget("GETTEXT",window,name,width)
```

Argument	Description
name	Name associated with the text box input UI control. Text entered in this input UI control is stored under this name.
width	Specifies the width of the text box. *width* is any positive integer. The value for *width* is approximately equal to the number of characters (in the default browser font) visible across the box at one time.

Notes

- The name of a text entry input UI control:
  - Must be a unique name throughout the pageset
  - Can contain letters and numbers, but must begin with a letter
  - Can contain no spaces or punctuation except for an underscore
- There are currently no restrictions on the number of characters that can be typed into a text entry field.
- Do not enter the following text into a text entry field:
  - Unescaped double quotes
  - Any opening script tag such as < !-- or < SCRIPT >

Example

The following sample code creates a text entry input UI control. The caption text is created using a separate HTML code. Text entered into this input UI control is saved under the name PLATETEXT.

```
Enter text for your custom license plate:

<SCRIPT>document.write(ISS.BuildWidget("GETTEXT",window,
"PLATETEXT",8));</SCRIPT>
```

LinkToSubConfig

Usage

The LinkToSubConfig function changes the display page from one portion of a subconfigured system to another portion.

For example, a parent page might contain a link to a child page. When a user selects the link, the parent user interface is replaced with the child user interface. This only affects the user interface since in a subconfigured system, all parent and child data is always present.

Typically, you would use the SUBCONFIG_LINK OPT_SUBCONFIG_LINK arguments of the BuildTarget function instead of calling the LinkToSubConfig function directly.

Syntax

LinkToSubConfig("*TOP*");

LinkToSubConfig("*PARENT*");

LinkToSubConfig(*tableName*);

Argument	Description
tableName	Name of the relative instance that describes the subconfigured child pageset.

Example

The following sample code uses LinkToSubConfig to link from the current pageset to a parent pageset:

```
ISS.LinkToSubConfig("PARENT");
```

This sample code uses LinkToSubConfig to link from a parent pageset to a child pageset:

```
ISS.LinkToSubConfig("BURGER");
```

LoadFile

Usage Use the LoadFile function to load a display page file into a specified frame. Use this call instead of standard JavaScript calls such as location.href = and location.replace.

Syntax ISS.LoadFile(*frameName*,*fileName*[,*path*])

Argument	Description
frameName	Absolute path, in string form, to the frame where *fileName* should be loaded. Alternately, you can specify window (but not as a string) if loading into the current frame.
fileName	Name of the HTML file.
path	Optional. Full path to where *fileName* resides. Use one of the functions that returns directory paths (such as GetUIPath) to return the full path to the directory where the file resides. If no path is specified, *path* defaults to pg.

Examples The following sample code uses LoadFile to load the file dogs.htm (which resides in the pg directory) into the current frame.

```
<A HREF="#" onClick="ISS.LoadFile(window,'dogs.htm');
return false;">Click here to see dogs</A>
```

The following sample code loads the file dogs.htm (which resides in the ui directory) into the current frame:

```
<A HREF="#" onClick="ISS.LoadFile(window,'dogs.htm',
ISS.GetUIPath());return false;">Click here to see dogs</A>
```

The following sample code loads the file dogs.htm (which resides in the ui directory) into a particular frame called descArea:

```
<A HREF="#"
onClick="ISS.LoadFile(ISS.GetPagesetDisplayArea()+'.descArea','d
ogs.htm',ISS.GetUIPath());return false;">Click here to see dogs</
A>
```

LoadPageset

Usage

Use the LoadPageset function to send a user to another pageset from within a pageset without having to use the Contents List. LoadPageset can also be used to load a pageset with features that are not the default preselected features.

You can call LoadPageset from any frame within the application, and the function can execute anywhere a JavaScript function can be used.

Dynamic default strings are in the form "key1 = value1,key2 = value2". You can redefine the separator characters by editing the APP_STR_SEP_CHAR and APP_STR_EQUALS_CHAR variable definitions in the Application Configuration (\ui\app_config.js) file.

Syntax

LoadPageset(*pagesetName,dynDefStr,optArgObj*)

Argument	Description
pagesetName	The name of the pageset, specified in the format “project|pageset” (for example, Cars|Sedans).
dynDefStr	Optional. String that defines key-value pairs of input and value. The default format of *dynDefStr* is "key1 = value1,key2 = value2". Separator characters default to equal signs (=) and commas (,).
optArgObj	Optional. Reserved for future use.

Example

Example 1

```
<A HREF=”#” onClick="ISS.LoadPageset('PC_all',
'PROCESSOR=750');return false;">Go to Next Computer</A>
```

Instead of typing the key-value pairs in the function call, you can insert a reference to the column in the data model that lists the dynamic defaults by using a call to BuildTarget. Use this method if the dynamic defaults can vary depending on selections made on the page from which LoadPageset was called. The function call would look like this:

```
<A HREF=”#” onClick="ISS.LoadPageset('PC_all',
ISS.BuildTarget('TEXT',window,'DYNDEFS'));return false;">Go to
Next Computer</A>
```

Note that you can call BuildTarget only from a pageset display page.

Example 2

This example shows switching to project Accessories.

```
<A HREF="#"
onClick="ISS.LoadPageset('AccessorieslStorageDevices');return
false;">Go to Storage Devices</A>
```

LoadPagesetWithDynDefObj

Usage Use the LoadPagesetWithDynDefObj function to load a pageset with a dynamic default object. A dynamic default object represents a product with certain nondefault, preselected features.

LoadPagesetWithDynDefObj can be called from any frame within the application and can be executed anywhere a JavaScript function can be used. It is commonly called from the onClick event handler of a link, but it can also be referred to from an image map, set as the SUBMIT method of an embedded frame, or called from another user-defined JavaScript function.

LoadPagesetWithDynDefObj is used to load a particular pageset. The pageset ID is explicitly stated in the function call. To load a pageset with an ID that is determined by the valid configuration of another product, create a link output target using the BuildTarget function.

Syntax LoadPagesetWithDynDefObj(*url*[,*dynDefObj*,*optArgObj*])

Argument	Description
url	The name of the pageset, specified in the format "project|pageset" (for example, Cars|Sedans).
dynDefObj	Optional. Object that defines key-value pairs of input and value.
optArgObj	Optional. Reserved for future use.

Example

```
LoadPagesetWithDynDefObj("cats.htm",myDynDefObj);

LoadPagesetWithDynDefObj("cats.htm");
```

Usage Use the LoadPagesetWithDynDefObj function to load a pageset with a dynamic default object. A dynamic default object represents a product with certain nondefault, preselected features.

LoadPagesetWithDynDefObj

Contents List Functions 7

This chapter discusses several functions that allow you to customize the Contents List. The Contents List can appear in any visible frame of a browser-based application. You must set which frame should be used using the functions RegisterContentsListFrame and SetContentsListFrame. Once the contents list location has been set, calling ShowContentsList will display the Contents List in the designated frame.

RegisterContentsListFrame

Usage Use the RegisterContentsListFrame function to register which application frame displays the Contents List.

RegisterContentsListFrame can only be called at the start of a browser-based application. It is typically called from the onLoad event handler of the Application UI Definition file (\ui\ol_ui.htm).

NOTE: If there is no contents list, a blank must be registered. For example, ISS.RegisterContentsListFrame(“ “);

Syntax `RegisterContentsListFrame(`*`frameName`*`)`

Argument	Description
frameName	Full path to the frame in which the Contents List will appear. The path must be expressed relative to ISS.GetVisibleDisplayArea();.

Example Register the Contents List frame in the frameset defined inside the Application UI Definition file (\ui\ol_ui.htm):

```
<FRAMESET onLoad="ISS.RegisterContentsListFrame
(ISS.GetVisibleDisplayArea()+'.prodlist');">
```

See Also “FrameToOLString” on page 76.

“SetContentsListFrame” on page 135.

SetContentsListFrame

Usage Use the SetContentsListFrame function to redefine the Contents List display frame *while the application is running*.

The Contents List frame is originally set during application startup through a call to RegisterContentsListFrame. The SetContentsListFrame function redefines where the Contents List appears while the application runs.

SetContentsListFrame can be called anytime the application is running. It is usually called at the pageset level, from Pageset UI Registry file (\pg*pagesetID*_i.htm). You must give the full path to the frame, relative to the top of the application.

You must make sure that the target frame exists at the time the Contents List is loaded into it. Because each pageset can have its own set of frames, it is easy to reference a frame that no longer exists or does not yet exist.

Syntax SetContentsListFrame(*frameName*)

Argument	Description
frameName	Full path to the frame in which the Contents List will appear. The path must be expressed relative to ISS.GetDisplayArea();.

Example The following sample code uses SetContentsListFrame to reset the Contents List location to a frame named uidata, located inside a pageset frameset. The two examples are equivalent, but the first example is preferred.

```
ISS.SetContentsListFrame(ISS.GetPagesetDisplayArea()+".uidata")
```

or

```
ISS.SetContentsListFrame(ISS.GetVisibleDisplayArea()+".mainArea.
uidata")
```

See Also "FrameToOLString" on page 76.

"RegisterContentsListFrame" on page 134.

ShowContentsList

Usage Use the ShowContentsList function to show the Contents List in a previously registered frame.

ShowContentsList can be called from any frame within the application and it can be executed anywhere a JavaScript function can be used. It is commonly called from the onClick event handler of a link, but it can also be referred to from an image map, set as the SUBMIT method of an embedded frame, or called from a custom JavaScript function.

ShowContentsList displays the Contents List in the frame defined by either RegisterContentsListFrame or SetContentsListFrame, whichever is called last.

Syntax ShowContentsList()

ShowContentsList(*contentsListData*)

Argument	Description
contentsListData	Optional name of an HTML file containing Contents List data.

Example

```
ISS.ShowContentsList("contentsListdata1.htm");
```

The *contentsListData* argument is optional. Omitting it loads the default Contents List:

```
<A HREF="#" onClick="ISS.ShowContentsList();
return false;">Show Contents List</A>
```

See Also "RegisterContentsListFrame" on page 134.

Callout Point Functions 8

The browser-based engine observes a well-defined set of rules for building and processing configurations and orders. You may want to change the standard behavior of the engine running the browser-based application. You can attach custom code to callout points to customize the runtime experience of your application.

Callout point functions often require an understanding of the way that data flows through a browser-based application. Some callout points have arguments that provide data from the runtime engine. This data is strictly read-only and should never be overwritten under any circumstances.

These callout points are custom code written in JavaScript. The custom code lives in the customCode.js file found in the custom directory. Any routine found inside the customCode.js file is executed during runtime at a particular point in the browser-based engine code.

Using the set of override functions, you can override specific callout points at the pageset level. This allows you to custom tailor callout points to specific pagesets.

ClearAllOverrideFunctions

Usage Use the ClearAllOverrideFunctions function to reset all pageset-level custom functions and return them to their original states.

Because ClearAllOverrideFunctions is always called when a Pageset UI Registry file (\pg*pagesetID_i*.htm) loads, all pageset-level custom functions are cleared when a new pageset loads.

Syntax ClearAllOverrideFunctions()

Example

```
ISS.ClearOverrideFunctions();
```

ClearOverrideFunction

Usage Use the ClearOverrideFunction function after a custom function override to reset the custom function to the original function and to return the application to its original state.

Syntax ClearOverrideFunction(*fnName*)

Argument	Description
fnName	Name of the overridden function you want to reset.

Example

```
ISS.ClearOverrideFunction("COP_BeforeConfiguration");
```

OverrideFunction

Usage Use the OverrideFunction function to override a specified function.

You can use OverrideFunction to change a default callout point at the pageset level instead of the application level. Override functions are automatically cleared when a Pageset UI Registry file (\pg*pagesetID*_i.htm) loads.

Syntax OverrideFunction(*fnName,fnPointer*)

Argument	Description
fnName	Name of the standard function you want to override.
fnPointer	Pointer to the custom function to be called in place of the original function.

Example The following sample code shows how to use OverrideFunction in a Pageset UI Registry file (\pg*pagesetID*_i.htm) to call the custom function MY_COP_BeforeConfiguration in place of the standard function COP_BeforeConfiguration.

```
function MY_COP_BeforeConfiguration(privateArg1,
privateArg2,instanceName,configData,featureData,
privateArg3,inputState, privateArg4, privateArg5)

{alert("my_function");

}

ISS.OverrideFunction("COP_BeforeConfiguration",
MY_COP_BeforeConfiguration);
```

COP_AppDataVersionCheck

Usage COP_AppDataVersionCheck is called when a user tries to reopen an item from a saved order that is out of date. The application compares the APP_DATA_VERSION variable in the current application to the variable set for the saved configuration.

Use the COP_AppDataVersionCheck function to determine the action that occurs when a user tries to link back from an order to a configured item that is out of date in a browser-based application.

The COP_AppDataVersionCheck function must return true or false. If the function returns true, the pageset continues loading. If the function returns false, the pageset does not.

Set the data version of an application in the Application Configuration file (\custom\app_config.js) using the variable APP_DATA_VERSION.

Syntax COP_AppDataVersionCheck(*argObj*)

Argument	Description
argObj	Object, with the following attribute: *argObj.appDataVersion:* data version of the application in which the item was originally created.

COP_BeforeConfiguration

Usage The browser-based engine calls COP_BeforeConfiguration prior to the configuration evaluation. After this function executes, the engine uses input UI control values to evaluate engine results.

Syntax COP_BeforeConfiguration(*privateArg1,privateArg2,instanceName,configData, featureData, privateArg3,inputState,privateArg4,privateArg5*)

Argument	Description
instanceName	Name of the instance currently being evaluated.
configData	Configuration table array object. See "ConfigTableArray_Obj" on page 165.
featureData	Feature table array object. See "FeatureTableArray_Obj" on page 167.
inputState	List of Feature tables for which the selection has changed since the last engine run. See "InputState_Obj" on page 168.

See Also "Example 4: Add Custom Behavior with a Callout Point" on page 179

COP_BeforeDisplayPriceString

Usage This allows you to manipulate the HTML string before it appears in the Pricing window.

Whatever changes you make to the original string 's' must be returned by this function. In other words, the end of this function must contain a 'return s' or whatever variable was chosen to hold the HTML string.

Syntax COP_BeforeDisplayPriceString(*PriceObj, s, cols, font, body, title, table, close*)

Argument	Description
PriceObj	A hierarchical price object sent from the server. This object may contain child objects if available.
s	A string containing the entire HTML output for the pop-up window. This includes everything from < HTML > to < /HTML > that gets assembled through the standard code or through the callout point.
cols	An array of strings that contain the data to be displayed, in the order in which they appear.
font	A string that controls the font display. < FONT > tag.
body	A string containing the attributes for the < BODY > tag, such as background color.
title	A string that will display in the pop-up window title area. < TITLE > tag.
table	A string that contains attributes for the < TABLE > tag. All data for the pricing window appears in a table.
close	A string that contains the text displayed in the optional Close Window link.

Example The following sample code uses COP_BeforeDisplayPriceString to replace the Close Window text with an image.

```
   function COP_BeforeDisplayPriceString(PriceObj, s, cols, font,
body, title, table, close) {

      return s.replace(/Close Window/,"<img
src='"+GetUIPath()+"close.gif' border=0>");

   }
```

The following sample code displays the child pricing objects in indented format. The default pricing display is just the parent object.

Sample output:

ACME Sedan	2001 ACME Sedan V8	$9,000
Automatic Transmission	5-speed Automatic Trans	$1000
Air Conditioning	Automatic Air Conditioning	$900
Stereo Upgrade	8-Speaker Premium Sound	$500
CD Changer	Trunk-mounted CD Changer	$500

The ACME Sedan is the parent object. The Automatic Transmission, Air Conditioning and Stereo Upgrade are children (and they are indented). The CD Changer is a child of the stereo upgrade so it is further indented. For more information on parent/child modeling, refer to the Referencing Other Siebel Data chapter in *Siebel Interactive Designer Administration Guide*.

The sample code, which should be placed at the end of custom/customCode.js:

```
function COP_BeforeDisplayPriceString(PriceObj, s, cols, font, body,
title, table, close) {

if (typeof PriceObj != "undefined" && PriceObj != null) {

   var s = "<html><head><title>"+title+"</title></head><body
"+body+">";

   s += "<font "+font+">";

   s += "<table "+table+">";
```

```
      // display all rows of data.
   s += DisplayRow(PriceObj, cols, font);


   s += "</table>";
   // only display the close link if it is defined and if in a window
(not frame)
   if (close != "" && dispFrame == false) s += "<br><br><a
href='javascript:void(0)' onclick=self.close()>"+close+"</a>";
   s += "</font>";
   s += "</body></html>";
   return s;
}
}


// DisplayRow displays the PriceObj data as specified in cols. The
depth determines how far to indent the description.
// Recursively displays all of the children and grandchildren.
function DisplayRow(PriceObj, cols, font, depth) {
var s = "";         // return string
var length;
var indent = "";
if (depth == null) {depth = 0};
var children = PriceObj.GetChildren();


s += "<tr>";
```

```
if (cols.length>0) {
   for (var i=0;i<cols.length;i++) {
      if (typeof PriceObj.GetField(cols[i]) != "undefined") {

   // indent only the first field
   if (i==0 && depth > 0) {
      for (var d=0;d<depth;d++) {
         indent += "   "
      }
   } else {
      indent = "";
   }

      s += "<td><font "+font+">"+indent+PriceObj.GetField(cols[i])
            +"</font></td>";
   } else {
      s += "<td> </td>";
   }
}
}
s += "</tr>";

// check for children objects, if exist, display them.
length = children.length;
if (length > 0) {
```

```
   for (var j=0;j<length;j++) {
      s += DisplayRow(children[j],cols,font,++depth);
      depth--;
   }
}
return s;
}
```

COP_InvalidItemAdded

Usage The COP_InvalidItemAdded function is called when a user configures an invalid item and tries to add it to the order.

Syntax COP_InvalidItemAdded()

Example

```
function COP_InvalidItemAdded()

{

alert("I'm sorry, the item that you are trying to order is not
configured properly.\n  Please try again.");

}
```

COP_PagesetVersionCheck

Usage COP_PagesetVersionCheck is called when a user tries to reopen an item from a saved order that is out of date. The application compares the current pageset version to the pageset version of the saved configuration. Set the data version of a pageset in the Pageset UI Registry file (\pg*pagesetID*_i.htm) using the InitPagesetVersion function.

Use the COP_PagesetVersionCheck function to determine the action that occurs when a user tries to link back from an order to a pageset that is out of date in a browser-based application. If the COP_PagesetVersionCheck function returns TRUE, the pageset continues loading. If the COP_PagesetVersionCheck function returns FALSE, the pageset does not load.

Syntax COP_PagesetVersionCheck(*argObj*)

Argument	Description
argObj	Object, with the following attribute: *argObj.linkbackVersion*, which is the data version of the application in which the item was created.

Example

```
function COP_PagesetDataVersionCheck(argObj)

{

// Checks to see how out of date the item is. If it is too
// far out of date, it won't allow the item to be brought back
// into the application. Otherwise, it gives the user an option.

   var retVal = null;

   if(parseFloat(argObj["linkbackVersion"] )< 3.0) {

      retVal = false;

   }

   else {

   retVal = confirm("The item you are trying to edit may be out of
date.\n  Do you wish to continue?");

   }

   return retVal;

}
```

COP_ValidItemAdded

Usage The COP_ValidItemAdded function is called every time a user attempts to add a valid configuration to the order. COP_ValidItemAdded returns a Boolean value indicating whether the item should be added to the order (true) or not (false).

Syntax COP_ValidItemAdded()

Example

```
function COP_ValidItemAdded()

{

   var userResponse = confirm("Would you like to add this
item to the order?");

   return userResponse;

}
```

InitApp

Usage The InitApp function is called when the application starts. You can use the InitApp function to specify the actions that occur when the application loads.

InitApp must be defined for a browser-based application in the file customCode.js in the custom directory. The default function included in the application template shows only the Contents List. No default startup behavior is defined, so if no actions are specified by InitApp, no actions are performed.

InitApp should only contain calls to display in the UI if LOAD_UI_ON_STARTUP = TRUE. This function is called when the code loads. In the case of Siebel Integrated mode, the UI is not loaded when the code loads.

Syntax InitApp()

Example In the following example, the application starts with the Contents List displayed and the dogs pageset loaded and displayed:

```
function InitApp() {

 if
(ISS.GetConfigVarWithDefault("APP","LOAD_UI_ON_STARTUP",true)) {

  ISS.ShowContentsList();

  ISS.LoadPageset("dogs");

 }

}
```

ORP_DisplayPrice

Usage ORP_DisplayPrice allows you to write custom code to bypass the entire code for GetPrice. The user is passed the same information, including the price object, that is returned from the server and all the configuration variables.

Syntax ORP_DisplayPrice(PriceObj, cols, title, body, font, table, close, hPriceWin, 'PriceWin', winAttr)

Argument	Description
PriceObj	The price object.
cols	An array of strings that contain the data to be displayed, in the order they appear in.
title	A string that will display in the pop-up window title area. < TITLE > tag.
body	A string containing the attributes for the < BODY > tag.
font	A string that controls the font display. < FONT > tag.
table	A string that contains attributes for the < TABLE > tag.
close	A string that contains what the optional Close Window link will say.
hPriceWin	Contains the pop-up window's handle so that you can access the same pop-up window that may have been opened previously.
'PriceWin'	A string that indicates the name of the pop-up window. Note that this is a string and not a variable.
winAttr	A string containing the window attributes for the pop-up window.

ORP_DisplayPriceString

Usage ORP_DisplayPriceString allows you to write custom code to bypass the entire code for GetPrice. This function allows you to override the final display of the data.

At the end of the this function, you must return the handle of the window that was opened. This function is useful if you want to override the default behavior of opening a pop-up window for displaying the price.

Syntax ORP_DisplayPriceString(PriceObj, s, hPriceWin, 'PriceWin', winAttr)

Argument	Description
PriceObj	The price object.
s	A string containing the entire HTML output for the pop-up window. This includes everything from < HTML > to < /HTML > that gets assembled through the standard code or through the callout point.
hPriceWin	Contains the pop-up window's handle so that you can access the same pop-up window that was opened previously.
'PriceWin'	A string that indicates the name of the pop-up window. Note that this is a string and not a variable.
winAttr	A string containing the window attributes for the pop-up window.

Data Access Functions 9

Data access functions provide structured access to complex data objects inside the browser-based engine. These functions are provided for application developers who need internal engine data for their custom code.

You can only call data access functions from within certain points of the application.

GetCurrInstance

Usage Use the GetCurrInstance function to get the instance name for the product currently being displayed.

A subconfigured system consists of a parent and one or more children. Each of the nodes in the system (including the parent) is an "instance." Occasionally you will need to know the name of the instance currently being considered. Calling this function returns the name of the current instance.

The value returned by GetCurrInstance corresponds to the currently displayed pageset. For example, if you are configuring a rack that contains four computers (all defined in the same pageset), the current instance of the computer would be the one you are configuring at that moment (such as TOP:COMPUTER2).

For children, the instance name is derived from the name of the Feature table in the parent pageset that defines the subconfiguration relationship.

If you are at the top-level configuration, GetCurrInstance returns TOP.

Call this function while a product is actively being configured. Do not call this function when no product has been selected (for example, right after the application starts).

Syntax GetCurrInstance()

GetFeatureData

Usage Use the GetFeatureData function to return the set of Feature tables from the current pageset.

In most instances where you would consider calling this function, the Feature table data is accessible through another mechanism. For example, access to feature data is sometimes useful at COP_BeforeConfiguration. However, the feature data was explicitly presented by an argument.

You should call this function while a product is actively being configured. Do not call this function when no product has been selected (for example, right after the application starts).

Syntax GetFeatureData()

Return Value `FeatureTableArray_Obj`

GetInputState

Usage Use the GetInputState function to return an input state object.

This object contains the current input state (the set of Feature table selections). This roughly corresponds to the current set of interface UI control selections. Note that in some instances, there are more Feature tables in the input state than there are UI controls in the interface.

In most instances where you would consider calling this function, the input state is accessible through another mechanism. For example, access to the input state is sometimes useful at COP_BeforeConfiguration. However, the input state is explicitly presented by an argument.

You should call this function while a product is actively being configured. Do not call this function when no product has been selected (for example, right after the application starts).

Syntax GetInputState()

Return Value InputState_Obj

GetInputValue

Usage Use the GetInputValue function to return the *index* of the current selection for a Feature table in the input state.

For example, in a list box UI control, the items are pulled from the description field in a Feature table. When a user selects a description, they are really selecting the corresponding row in the Feature table. This function returns the index that identifies the row.

The index is different than the code. The index is the row number. The index is 0-based.

Function is most often used writing custom code for creating custom input UI controls.

Call this function while a product is actively being configured. Do not call this function when no product has been selected (for example, right after the application starts).

Syntax GetInputValue(*featureTableName*)

Argument	Description
featureTableName	Name of the Feature table.

Example If the SIZE input UI control choices are Small, Medium, and Large, and the current UI control selection is Large, the following sample code returns 2:

```
var selection = ISS.GetInputValue("SIZE");
```

See Also "SetInputValue" on page 161.

GetResultsValue

Usage Use the GetResultsValue function to retrieve a single value from the results set generated by the engine.

This function is called by the BuildTarget function. Consider calling this function directly if you are developing a custom output target or trying to access a value from a display page.

Call this function while a product is actively being configured. Do not call this function when no product has been selected (for example, right after the application starts).

Syntax GetResultsValue(*targetName*)

Argument	Description
targetName	Name of a data element that can be used as an output target (for example, Configuration table column name, Feature table name, Feature table column name, and so on.)

Example The following sample code returns the price of the current item, where PRICE is a Configuration table column name:

```
var price = ISS.GetResultsValue("PRICE");
```

See Also "Example 3: Create a Custom Output Target" on page 178

SetInputValue

Usage Use the SetInputValue function inside custom code. The function changes the input state and then it causes the engine to run immediately.

The SetInputValue function is called by the BuildWidget function and should rarely be called directly. Call this function only if you are developing a custom input UI control.

Call this function while a product is actively being configured. Do not call this function when no product has been selected (for example, right after the application starts).

Syntax SetInputValue(*table,selIndex*)

Argument	Description
table	name of Feature table that corresponds to the input UI control.
selIndex	index that indicate which Feature table row has been selected.

Example When the BuildWidget function for a check box executes, it produces the following HTML and JavaScript:

```
<INPUT type=checkbox name="CABLE_INCL" language="JavaScript"
onClick="ISS.SetInputValue('CABLE_INCL',(checked) ? 1:0)">
```

When a user selects the check box, the SetInputValue function is called. The SetInputValue function then changes the value of the CABLE_INCL Feature table selection and runs the engine.

See Also "GetInputValue" on page 159.

PostThis()

Usage Use the PostThis() function to post data in the Siebel application database.

Syntax PostThis()

Argument	Description

Example

```
PostThis();
```

Data Objects 10

Browser-based application data is organized and exposed through structured data objects.

You can control many of the data objects described in this section with data access functions. Whenever possible, use these data accessor functions instead of directly manipulating the data object itself.

ConfigTable_Obj

Usage The ConfigTable object contains data for one configuration table.

Methods

Method	Description
GetBody()	Returns the body information.
GetColType()	Returns an array of column type information. Column types indicate whether a particular column in the table is an output, input, or subtable column.
GetException()	Returns an array representing the exceptions portion of the table.
GetHeader()	Returns an array of table column names.

Example See “Example 4: Add Custom Behavior with a Callout Point” on page 179.

See Also “ConfigTableArray_Obj” on page 165.

ConfigTableArray_Obj

Usage The ConfigTableArray object stores an array of Configuration table objects.

Methods

Method	Description
GetAllTables()	Returns an array of all Configuration tables.
GetTable(*tableName*)	Returns the ConfigTable_Obj for the *tableName* requested. Returns null if the specified table does not exist.

Example See "Example 4: Add Custom Behavior with a Callout Point" on page 179.

See Also "ConfigTable_Obj" on page 164.

FeatureTable_Obj

Usage The FeatureTable object contains Feature table data.

Notes Each row in a Feature table is represented by an object that contains one field for each column in the table. The field names are identical to column names. Feature table rows are indexed starting with 0.

Method

Method	Description
GetRows()	Returns an array of rows from the Feature table.
GetRow(*index*)	Returns the specified row from the Feature table. Returns null if the row does not exist.
GetDefaultRow()	Returns the default row.
GetDefaultRowNum()	Returns the index number of the default row.
GetRowCode(*rowNum*)	Returns the value from the CODE column for the specified row in the table.
GetRowDesc(*rowNum*)	Returns the value from the DESC column for the specified row in the table.
GetRowNumFromCode(*CodeValue*)	Returns the row number for a given CODE value. Returns null if the cell does not exist.
GetTableCell(*rowNum*,*colName*)	Given a row number and a column name, returns the value of the table cell. Returns null if the cell does not exist.

Example See "Example 4: Add Custom Behavior with a Callout Point" on page 179.

See Also "FeatureTableArray_Obj" on page 167.

FeatureTableArray_Obj

Usage The FeatureTableArray object stores an array of FeatureTable objects.

Methods

Method	Description
GetAllTables()	Returns an array of all Feature tables.
GetTable(*tableName*)	Returns the FeatureTable_Obj for the *tableName* requested. Returns null if the specified table does not exist.

Example See "Example 4: Add Custom Behavior with a Callout Point" on page 179.

See Also "FeatureTable_Obj" on page 166.

InputState_Obj

Usage The InputState object contains all input selections.

Methods

Method	Description
GetValue(*tableName*,*instanceName*)	Gets the value of an input. Returns row index for inputs associated with Feature tables. Returns literal value for inputs not associated with a Feature table.
SetValue(*tableName*,*instanceName*,*value*)	Sets the value of an input. Sets row index for inputs associated with Feature tables. Sets literal value for inputs not associated with a Feature table.

Example See "Example 4: Add Custom Behavior with a Callout Point" on page 179.

See Also "COP_BeforeConfiguration" on page 142.

Label_Obj

Usage

Use the Label_Obj object to provide access to the data in a Feature table.

The object has two methods, GetLabelName and GetNumLabels, which you can use to populate an input UI control.

Use the Label_Obj function only when writing custom code for creating custom input UI controls and output targets.

Syntax

new Label_Obj(*featureTableName*)

Argument	Description
featureTableName	Name of a Feature table corresponding to the UI control.

Methods

Method	Description
GetLabelName(index,colName)	For a description, see "Label_Obj.GetLabelName" on page 170
GetNumLabels()	For a description, see "Label_Obj.GetNumLabels" on page 171

Example

See "Example 1: Create a Custom Input UI Control" on page 174.

Label_Obj.GetLabelName

Usage Use the GetLabelName method to return a value from a Feature table.

You can use GetLabelName to populate an input UI control. Given an index and column name, the function returns the corresponding cell value.

You do not need to state the Feature table name here, as it was specified when the Label_Obj was created. You are accessing *Feature_table[row_index].column_name*.

Use this function only when writing custom code for creating custom input UI controls and output targets.

Syntax label.GetLabelName(*index,colName*)

Argument	Description
index	Integer that represents the index number of the desired row in the Feature table that corresponds to this label.
colName	Optional. Name of a Feature table column. If none specified, the name defaults to DESC.

Example See "Example 1: Create a Custom Input UI Control" on page 174.

Label_Obj.GetNumLabels

Usage Use the GetNumLabels function to return the number of rows in a Feature table.

You can use this function to iterate through possible selections or verify that a given index is valid for a Feature table. Use this function only when writing custom code for creating custom input UI controls and output targets.

Syntax label.GetNumLabels()

Example See "Example 1: Create a Custom Input UI Control" on page 174.

Label_Obj.GetNumLabels

API Examples 11

This chapter contains examples that use various API functions to perform common tasks.

Example 1: Create a Custom Input UI Control

The code in the following example defines a graphical radio button input UI control that uses images instead of standard radio buttons to indicate whether the input UI control is selected. As with a radio button, the state of this input UI control changes when users click the radio button image.

Some definitions in this example refer to the following:

winObj	Window that the input UI control is in.
featureTableName	Name of the Feature table from which to build the UI control.
index	Row of the Feature table you are currently accessing.
pictOn	Column name in Feature table which contains path to "selected" image for this row.
pictOff	Column name in Feature table which contains path to "unselected" image for this row.

```
function BuildImageWidget(winObj,featureTableName,index, pictON,
pictOFF)

{

   ISS.RegisterUIElement(featureTableName, winObj, true);


   var bPict = false;

   var srcON, srcOFF;

   var label = new ISS.Label_Obj(featureTableName);

   if (index >= label.GetNumLabels()) { return ""; }


   var srcON = label.GetLabelName(index,pictON);

   var srcOFF = label.GetLabelName(index,pictOFF);
```

```
   var presel = ISS.GetInputValue(featureTableName);
   if (presel == null && typeof presel == "undefined") { return ""; }

   var src;
   if (presel==index) {
      src = srcON;
   } else {
      src = srcOFF;
   }

   var s = '<A HREF=”#” ' ;
   s += ' onClick="ISS.SetInputValue(';
   s += '\''+featureTableName+'\','+index;
   s += ');return false;">';
   s += '<IMG align="absmiddle" border = 0 src="'+src+
'"</IMG></a>';
   return s;
}
```

Example 2: Define a Pageset Layout

The code in the following example defines a pageset user interface inside the files xxx_i.htm and oc_default_ui.htm, both of which would reside in the application pg directory. The user interface defined in this example has the following characteristics:

- The pageset display area is ISS.GetPagesetDisplayArea() + ".oc_frame".
- The pageset display area is subdivided based on the frameset defined in oc_default_ui.htm, which defines three frames: oc_frame1, oc_frame2, and oc_frame3.
- The HTML display page named xxx_1.htm is mapped into oc_frame1.
- The HTML display page named xxx_2a.htm is mapped into oc_frame2.
- The page xxx_1.htm displays before the engine executes.
- Exception messages display in the frame oc_frame2.
- The HTML display page named xxx_2b.htm can be mapped into oc_frame2 by using the LoadFile function.
- The Contents List displays in oc_frame3.

Contents of xxx_i.htm

```
<HTML>

<HEAD>

<TITLE>Interface Template</TITLE>

<SCRIPT src="../jd/header.js"></SCRIPT>

<SCRIPT language=javascript>

ISS.StartUIInfo('xxx');

ISS.RegisterFrameSet("oc_default_ui", "oc_default_ui.htm,
ISS.GetPagesetDisplayArea());

ISS.RegisterPageLocation("xxx_1.htm",
ISS.GetPagesetDisplayArea()+".oc_frame1");
```

```
ISS.RegisterPageLocation("xxx_2a.htm",
ISSGetPagesetDisplayArea()+".oc_frame2","true");

ISS.RegisterPageLocation("xxx_2b.htm",
ISSGetPagesetDisplayArea()+".oc_frame2","false");

ISS.RegisterPriorityPages("xxx_1.htm");

ISS.RegisterExceptionFrames(
ISSGetPagesetDisplayArea()+".oc_frame2");

ISS.SetContentsListFrame(ISSGetPagesetDisplayArea()+".oc_frame3");

</SCRIPT>

</HEAD>

<BODY onLoad="ISS.EndUIInfo('xxx')"></BODY>

</HTML>
```

Contents of oc_default_ui.htm

```
<HTML>

<HEAD><TITLE>CDA(tm): 7.0 UI Definition</TITLE></HEAD>

<SCRIPT></SCRIPT><SCRIPT src="../jd/header.js"></SCRIPT>

<FRAMESET cols = "200,200,*" border=0 frameborder=0 framespacing=0
onLoad="ISS.SetLoaded('oc_default_ui');">

   <FRAME marginwidth=0 marginheight=0 src="javascript:''"
name="oc_frame1" scrolling="auto">

   <FRAME marginwidth=0 marginheight=0 src="javascript:''"
name="oc_frame2" scrolling="auto">

   <FRAME marginwidth=0 marginheight=0 src="javascript:''"
name="oc_frame3" scrolling="auto">

</FRAMESET>

<NOFRAMES><BODY></BODY></NOFRAMES>

</HTML>
```

Example 3: Create a Custom Output Target

The code in the following example defines a "button" output target. Clicking the HTML button defined by the output target launches a JavaScript alert window. The alert window contains a string that varies depending on the current configuration.

```
function BuildSecretButtonTarget(winObj,name)

{

   // Register that this element must be refreshed when
   // results change

   ISS.RegisterUIElement(name, winObj, true);

   var secretStr = ISS.GetResultsValue(name);

   if (secretStr == "") return "";


   var retStr = '<INPUT TYPE="button" value="My Secret"
onclick="alert(\''+secretStr+'\')">';

   return retStr;}
```

Example 4: Add Custom Behavior with a Callout Point

This example adds custom behavior to be executed before the configuration engine runs. Using values from the Feature tables for selected rows, an input selection value is calculated and set.

```
function COP_BeforeConfiguration(privateArg1,

privateArg2,instanceName,configData,featureData,

privateArg3,inputState,privateArg4,privateArg5) {


   // get the current selection for the packing method

   var pmVal=inputState.GetValue("PACK_METHOD",instanceName);


// if the packing method selection is to use cartons,

   // calculate and set the number of cartons

   if (featureData.GetTable("PACK_METHOD").GetRowCode(pmVal)==
"CARTON") {


      // get quantity of eggs

      var eggSel=inputState.GetValue("EGGS",instanceName);

      var total=featureData.GetTable("EGGS").GetTableCell(eggSel,
"QTY");


      // calculate how many cartons are needed

      var eggsPerCarton=
featureData.GetTable("PACK_METHOD").GetTableCell(pmVal,
"EGGS_PER_CONTAINER");

      var numCartons=parseInt(total/eggsPerCarton);

      if (total%eggsPerCarton!=0) numCartons++;
```

```
// set the selection of CARTONS to numCartons.
inputState.SetValue("CARTONS",instanceName,numCartons);
```

Example 5: Use the CDA Service Broker

This example shows how to use the CDA Service Broker method in the ISSCDA RT UI Service. The example includes the following three parts:

- Client JavaScript
- Client HTML
- Business service script

Client JavaScript

This is an example of the client JavaScript.

```
<script>

//Include the propset.js file in the SWE scripts directory if you
are using the CDAReturnHandler parameter option

// Example function shows how to use the CDAServiceBroker

function PostThis(method, varName, varVal){

   var argObj = new Array();

   // Defines the business service that you want CDAServiceBroker to
   call

   argObj["CDAExternalServiceName"] = "CDA Broker Test";

   if(typeof method != 'undefined')

      argObj["CDAExternalMethodName"] = method;

   if(typeof varName != 'undefined' && typeof varVal != 'undefined')

      argObj[varName]=varVal;

   argObj["target"] = OL.FrameToString(OL)+"."+"dataFrame";

   OL.SendSelectionInformationToServer("ISSCDA RT UI Service",
   "CDAServiceBroker", "prodStr", true, argObj);

}
```

```
// Example event handler to demonstrate how the CDAServiceBroker can
be used

function alertEvent(ps)

{

   if (ps != null)

   {

   for (bFirst = true; (arg = ps.EnumProperties (bFirst)) != null;
   bFirst = false)

      {

         value = ps.GetProperty (arg);

         alert(arg + "=" + value);

      }

   }

   else

   {

   alert("Empty PropertySet in function alertEvent");

   }

} </script>
```

Client HTML

The following HTML code is an example of the body (<BODY>) section of the output layout file.

```
<!—Client request requires no confirmation from the BC -->

<p><a href="javascript:''" onClick="PostThis('NoInput'); return
false;">No return specified</A>

<!—Client request that requires no confirmation from the business
service, but the BC switches view ->

<p><a href="javascript:''" onClick="PostThis('NoInputGotoView');
return false;">No return specified but the server will switch view</
A>



<!—Client request that requires a confirmation from the business
service that calls an event handler ->

<p><a href="javascript:''"
onClick="PostThis('OutputEventHandler','CDAReturnHandler',OL.Frame
ToString(window)+'.alertEvent');return false;">Using eventHandler
for the propertyset</A>



<!—Client request that requires a confirmation from the business
service that sets the default property set values ->

<p><a href="javascript:''"
onClick="PostThis('OuputtoDefaultPropertySet'); return
false;">Default Output</A>



<!—Client request that requires a confirmation from the business
service that sets a specific property set values->

<p><a
href="javascript:''"onClick="PostThis('DefinedOutput','CDAReturnVa
lue', 'myoutput'); return false;">defined output with myoutput</A>

<p>
```

Business Service Script Called by CDAServiceBroker

This is an example business service script (CDA Broker Test) called by CDAServiceBroker. The example shows how to set up the response messages for the client.

```
function Service_PreInvokeMethod(methodName, inputArg, outputArg)

{

   var outstr = "<html><head></head><body>";

   var alertstr="from method: "+methodName;

   outstr+="<script>alert(\'"+alertstr+"\');</script>";

   outstr+="</body></html>";


   if(methodName == "NoInput") {

      return (CancelOperation);

   }

   else if(methodName == "NoInputGotoView") {

      TheApplication().GotoView("Quote List View");

   }

   else if(methodName == "OuputtoDefaultPropertySet") {

         outputArg.SetProperty("CDAReturnValue", outstr);

      return (CancelOperation);

   }

   else if(methodName == "DefinedOutput") {

         outputArg.SetProperty("myoutput", outstr);

      return (CancelOperation);

   }

   else if(methodName == "OutputEventHandler") {
```

```
            outputArg.SetProperty("method", methodName);
            outputArg.SetProperty("method2", methodName);
         return (CancelOperation);
      }
      else {
         return (ContinueOperation);
      }
   }
```

Example 5: Use the CDA Service Broker

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W