Contents

Chapter 1: What’s New in This Release

Chapter 2: Overview of the JavaScript API
Overview of Customizing Buttons 7
   Creating Custom Buttons 7
   Examples of Uses for Custom Buttons 7
   Hiding and Disabling Buttons 8
   Changing the Look and Feel of a Button 8
   Changing the Behavior of Buttons 8
   Error Handling 9
Contexts in Which You Can Customize Buttons 9

Chapter 3: Getting Started with the JavaScript API
Overview of Using Custom JavaScript Code 11
Privileges Required 11
Uploading JavaScript Libraries 11
Guidelines for Uploading Client-Side Extensions 12
Managing HTML Head Additions 13
Guidelines for Managing HTML Head Additions 14
Security Considerations 14

Chapter 4: JavaScript API Reference
Classes Exposed 15
   TitleBar Class 15
   Button Class 16
   Context Class 16
JavaScript API 16
   Chaining of Methods 16
Methods for the oraclecrmod Object 17
Methods for the TitleBar Object 19
Contents

Methods for the Button Object  19
  Defining an Event Handler for a Button  23
  Identifying the IDs of Buttons and TitleBars  23
Methods for Getting and Setting Screen Values  24
  Finding Field Names  25
  Guidelines for Setting Screen Values  26
Methods for the Context Object  26
Methods for CRUD Operations  28
  Parameters and Return Values for CRUD Methods  30
  Finding Record Type Names  31
  Guidelines for Performing CRUD Operations  31
  User-Defined Callback Function  32
  Helper Functions for Callback Functions  32
  Sample Code for Callback Function  33
Errors and Error Handling  33

Appendix A: JavaScript API Code Samples
  Code Sample for Creating a Custom Button for Validation  37
  Code Sample for a Custom Button That Creates a Record  38
  Code Sample for a Custom Button That Updates a Record  39
  Code Sample for a Custom Button That Gets a Shipping Address to Pass to an External Site  41
  Code Sample for Changing the Behavior of a Save Button  42

Index
What’s New in This Release

What’s New in Oracle CRM On Demand JavaScript API Developer’s Guide, Release 26

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

Table 1. New Product Features in Oracle CRM On Demand JavaScript API Developer’s Guide, Release 26

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Methods for CRUD Operations” on page 28</td>
<td>Modified topic. The description of the readRecord method in Table 9 on page 29 has been updated.</td>
</tr>
<tr>
<td>“Code Sample for a Custom Button That Updates a Record” on page 39</td>
<td>New topic. A code sample for creating a button that updates a record has been added.</td>
</tr>
</tbody>
</table>
2 Overview of the JavaScript API

This chapter provides an overview of the public JavaScript application programming interface (API) available with Oracle CRM On Demand. It contains the following topics:

- Overview of Customizing Buttons
- Contexts in Which You Can Customize Buttons on page 9

Overview of Customizing Buttons

You can use the methods of the JavaScript API to customize buttons in the Oracle CRM On Demand UI as follows:

- To create custom buttons
- To hide and show buttons
- To disable and enable buttons
- To change the look and feel of preconfigured and custom buttons
- To change the behavior of preconfigured and custom buttons

More information about the various ways in which you can customize buttons is given in the following topics. For information about the types of pages on which you can customize buttons, see "Contexts in Which You Can Customize Buttons" on page 9.

Creating Custom Buttons

The JavaScript API provides methods to create new custom buttons, for example, on record Detail pages, on related information applets, or on applets on My Homepage. To create custom buttons, you can use the createButton() method.

To retrieve both preconfigured buttons and custom buttons, you use the getButton() method. For more information about these methods, see "Methods for the oraclecrmod Object" on page 17.

Examples of Uses for Custom Buttons

Some examples of the uses for custom buttons are as follows:

- A custom button on Account record pages to validate the DUNS number
- A custom button to validate entered data and return error messages and details of fields that must be corrected
- A custom button to change the owner record or assign a book.
Hiding and Disabling Buttons

The JavaScript API provides methods to hide buttons and show buttons that were previously hidden. This feature is useful for situations when preconfigured buttons in Oracle CRM On Demand are not required by the user. You can hide buttons unconditionally or hide buttons based on some specified condition, for example, based on user roles.

You can use methods to disable buttons so that they are grayed out, and to enable buttons that were previously disabled. For information about all of these methods, see “Methods for the Button Object” on page 19.

Changing the Look and Feel of a Button

The JavaScript API provides methods to change the look and feel of a button. You can do the following:

- Set or change the display text for a button.
- Set or change the image for a button by providing a URL to the image.

You can use methods to get information about the text and image for a button. For information about these methods, see “Methods for the Button Object” on page 19.

Changing the Behavior of Buttons

The JavaScript API enables you to specify the onclick event handling for a custom button or modify the onclick event handling of a custom button or preconfigured button (such as New, Save, and Cancel), and thereby change its behavior.

You can change the click event handling of custom and preconfigured buttons to do the following:

- Set or retrieve on-screen values.
- Create, read, update, and delete (CRUD) off-screen records.

For information about registering and removing event handlers, see “Methods for the Button Object” on page 19 and “Defining an Event Handler for a Button” on page 23.

Getting and Setting On-Screen Values

You can use getter and setter methods to retrieve and set the values of fields on the current page when a user clicks a button. For information about these methods, see “Methods for Getting and Setting Screen Values” on page 24.

Creating, Reading, Updating, and Deleting Records

You can use methods to perform the following operations on off-screen records when you click a button:

- **Create record.** You can create a record and specify the record type, its fields, and their values.
Overview of the JavaScript API

Contexts in Which You Can Customize Buttons

- **Read record.** You can retrieve the fields from a record. You can specify the record type, record row ID, and a list of field names to be returned with their values.
- **Update record.** You can update a record identified with a specific row ID. You can specify the field names and values to update the record.
- **Delete record.** You can delete a record identified with a specific row ID.

For all of these create, read, update, and delete (CRUD) operations, you must have access to the record. If you do not, then the operation fails. You must implement a callback function to handle the results of the CRUD operation, when they become available. For information about the CRUD methods, see “Methods for CRUD Operations” on page 28.

Error Handling

For methods of the JavaScript APIs that return an object or value, null is returned when an issue occurs. However, the methods for CRUD operations are an exception as they return an error object.

When using the JavaScript API you must implement your own error handling, which gives you the flexibility to either show the error, or capture and handle it. For more information about error handling, see “Errors and Error Handling” on page 33.

Contexts in Which You Can Customize Buttons

Using the JavaScript API you can customize buttons in the following contexts:

- On the Homepage for a record type, such as Account, Contact, Lead, and so on
- On the Detail page for a record type
- On the Edit Page for a record type
- On related information (child) applets for record types, for example, the Account Related Information applet on contact records
- On applets on My Homepage
- On Administration pages
- On pop-up windows, including the following:
  - Quick create windows (new record form) available from the action bar
  - Single-association Lookup windows
  - Multiassociation Lookup windows
  - Multi-select picklist pop-up windows
  - Currency pop-up windows

Customizing buttons on any other page is not supported. Menu buttons are also not supported, and you cannot add custom buttons to the action bar.
Customizing buttons on both standard and customized page layouts is supported, and you can create custom buttons on customized page layouts. Different page layouts can have the same buttons with different behavior.

In the related information sections for records (child applets), button customizations will appear in both the tab view and the expanded view. For example, if you disable a button for a child applet, then it appears disabled in both the tab view of the related records and on the expanded view for the child record.

You can use methods to determine the context in which your code runs, including the page type and information about the currently logged-in user. For more information about these methods, see “Methods for the Context Object” on page 26.
This chapter provides an overview of how to get started with using the JavaScript API in Oracle CRM On Demand. It contains the following topics:

- Overview of Using Custom JavaScript Code
- Privileges Required
- Uploading JavaScript Libraries
- Guidelines for Uploading Client-Side Extensions on page 12
- Managing HTML Head Additions on page 13
- Guidelines for Managing HTML Head Additions on page 14
- Security Considerations on page 14

**Overview of Using Custom JavaScript Code**

To use custom JavaScript code you must create appropriate custom HTML head additions so that the custom code is executed in the Oracle CRM On Demand pages. You add the custom HTML head additions in the Application Customization section in the Oracle CRM On Demand UI. For more information, see “Managing HTML Head Additions” on page 13.

Your custom HTML head additions can reference JavaScript libraries or external JavaScript files to make their functions available. You upload JavaScript libraries as client-side extensions in the Application Customization section in the Oracle CRM On Demand UI. For more information, see “Uploading JavaScript Libraries” on page 11.

**Privileges Required**

To upload JavaScript libraries and manage custom HTML head tags, your user role must include the privilege: Upload Client Side Extensions and Manage Custom HTML Head Tag. If you do not have this privilege, then the links for Client Side Extensions and Custom HTML Head Tag are not available on the Application Customization page in the Oracle CRM On Demand UI. Contact Oracle Customer Care if you do not have the privilege.

**Uploading JavaScript Libraries**

You can upload JavaScript library files as client-side extensions in Oracle CRM On Demand. These files can include custom functions and global variables, as well as other JavaScript libraries that you require.
To upload a JavaScript library

1. In the upper-right corner of any page, click the Admin global link.
2. In the Application Customization section, click Application Customization.
3. In the Application Setup section, click Client Side Extensions.
4. Click New.
5. Enter the following details:
   - **File.** Click the paperclip icon to find the file that you want to upload.
   - **MIME Type.** (Optional) This field indicates the Internet media type. If you provide a value, then enter the following:
     ```text/javascript```
     When you refer to the client side extension file from a custom HTML head addition, you use a `<script>` element as in the following example:
     ```html
     <script src="url to the client side extension" type="text/javascript"></script>
     ```
   - **Name.** A name for the client side extension. This field is required.
   - **URL Name.** A meaningful name used in the URL that points to the client side extension. It is recommended that you keep this field as short as possible. This field changes only when a user edits it. Therefore, if you replace the uploaded file, or if you change the client-side extension name, then the URL Name field value is unchanged. As a result, you can update, maintain and deploy multiple versions of the client-side extension without having to reconfigure the HTML head additions. This field is required.
6. Click Save.

After you click save, the Relative URL and Full URL fields are generated for the client-side extension and displayed in the Client Side Extension list.

For more information about managing client-side extensions, see Oracle CRM On Demand Online Help.

Guidelines for Uploading Client-Side Extensions

The guidelines for uploading client-side extensions are as follows:

- **Keep all client-side extensions in as few files as possible.** This guideline improves performance. Preferably, refer to only one JavaScript file from custom HTML head additions.

- **Allow your browser to cache your client-side extensions.** If you allow your browser to cache the client-side extensions, then the content does not have to be downloaded from the Oracle CRM On Demand servers each time that a user loads the page. If the client-side extension changes, then you must clear your cache so that Oracle CRM On Demand loads the page containing the extension. By default, client-side extensions are cached for 30 days.
Managing HTML Head Additions

When you add custom HTML head additions, you specify the appropriate code in `<script>` elements to be included in the HTML `<head>` element of your pages. The `<script>` elements can contain your custom JavaScript code and can also reference JavaScript files uploaded through client-side extensions or external JavaScript files. HTML head additions apply to all pages in Oracle CRM On Demand. There is a limit of 50,000 characters to the code that you enter as HTML head additions.

On the Edit Custom HTML Head Tag page, a Preview button enables you to validate any changes that you have made. Depending on your browser settings, errors might be displayed if badly formed HTML or JavaScript is added.

You can disable the custom HTML head additions by appending `disableCustomJS=Y` as a parameter to the URL for a page. If you navigate away from the page by clicking another link, then the URL parameter is not retained. You must specify the URL parameter each time that you require it. The parameter is used only when the current user’s role includes the Upload Client Side Extensions and Manage Custom HTML Head Tag privilege.

**NOTE:** Any custom code is the responsibility of the person or persons who created the code. Oracle Customer Care does not support custom code or the contents of linked-in code. When adding custom code, keep in mind that Oracle supports only calls to published APIs for different versions of Oracle CRM On Demand.

**To add a custom HTML head addition**

1. In the upper-right corner of any page, click the Admin global link.
2. In the Application Customization section, click Application Customization.
3. In the Application Setup section, click Custom HTML Head Tag.
4. In the Custom HTML Head Tags Detail page, click Edit.
5. In the HTML Head Additions text box, enter the `<script>` elements that you require.
   - The `<script>` element can contain JavaScript code or can reference JavaScript files through the `src` attribute. The following is an example of a reference to a file uploaded as a client side extension:
     ```html
     <script type='text/javascript' src='../user/content/Test.js'></script>
     ```
   - The following is a reference to an external file:
     ```html
     <script type='text/javascript' src='http://www.external.com/js/Test.js'></script>
     ```
6. Click Preview to validate any changes you have made.
7. Click Save.

For more information about managing HTML head additions, see *Oracle CRM On Demand Online Help*. 
Guidelines for Managing HTML Head Additions

To minimize the time taken to load pages, reference JavaScript files, either as client-side extensions or external JavaScript files, to include your business logic. Using referenced JavaScript files also leverages the browser's cache to speed up the loading time for a page.

Security Considerations

Securing against Web browser-based attacks is a requirement when developing custom code and is the responsibility of the persons creating the custom code. Browser caches are only as secure as the computers or other devices that are browsing Oracle CRM On Demand.

The JavaScript API feature has been disabled for customer service representative (CSR) impersonation.
This chapter describes the public JavaScript application programming interfaces (API) available with Oracle CRM On Demand. It includes the following topics.

- Classes Exposed
- JavaScript API on page 16
- Methods for the oraclecrmod Object on page 17
- Methods for the TitleBar Object on page 19
- Methods for the Button Object on page 19
- Methods for Getting and Setting Screen Values on page 24
- Methods for the Context Object on page 26
- Methods for CRUD Operations on page 28
- Errors and Error Handling on page 33

Classes Exposed

The Oracle CRM On Demand JavaScript API exposes the classes described in the following topics. The top-level namespace is oraclecrmod.

**NOTE:** Although the JavaScript language does not have classes, it is common for JavaScript frameworks to simulate classes, and this approach has been taken with the Oracle CRM On Demand JavaScript API.

TitleBar Class

The TitleBar class represents the title bar on Oracle CRM On Demand pages and is a container for the buttons in Oracle CRM On Demand. Each TitleBar instance on a page has its own unique ID that is assigned by the Oracle CRM On Demand framework. You can identify the TitleBar’s ID from examining the id attribute of the TitleBar's Document Object Model (DOM) element, or by following the procedure described in “Identifying the IDs of Buttons and TitleBars” on page 23.

You can get the existing TitleBar instance through the TitleBar's ID. With the TitleBar instance, you can then call custom JavaScript code to create custom buttons on the TitleBar. See Table 3 for information about the API method used to get a TitleBar instance.

You cannot create a new title bar or hide an existing title bar, using JavaScript code.
**Button Class**

The Button class represents buttons in the Oracle CRM On Demand UI. Each Button instance must have a unique ID. Preconfigured buttons are assigned a unique ID by the Oracle CRM On Demand framework, and you must specify a unique ID for custom buttons that you create with JavaScript code. If no ID is supplied when creating a custom button, then an internal unique ID is assigned to the Button instance.

A TitleBar instance is not required to create a Button object. However, a button must be displayed inside a title bar, so the TitleBar instance is required when trying to display the button.

For most Edit pages, Oracle CRM On Demand has a title bar at the top of the form and also at the bottom of the form and the same named button, say New, appears on both the top and bottom title bars. However, the buttons are not linked and are treated as different buttons. So, for example, if you want to hide the New button, then you must hide the two buttons in each title bar explicitly.

See Table 3 and Table 4 for information about the API methods used to get Button IDs and create buttons, and Table 5 for information about the API methods that operate on Button instances.

**Context Class**

The Context class (oraclecrmod.ctx) represents a global object. The ctx object has attributes including those for the current object (record type), page type, and logged-in user that allow you to determine the context in which you run your code. For information about the attributes, see "Methods for the Context Object" on page 26, in particular, Table 8 on page 27.

Not all attributes are available on all pages. For example, the object attribute is not available on My Homepage, and Layout attributes are not available on association Lookup windows.

Some helper functions are available as alternatives to using attributes directly. For information about the helper methods, see Table 7 on page 27.

**JavaScript API**

The following topics describe the methods that are available through the JavaScript API. For each method the parameters and return types are listed with examples of the calling method. These topics also provide information usage information about the various methods.

**Chaining of Methods**

The oraclecrmod library uses chained API style for those methods that do not usually have a return value (for example, setter methods). You can chain many methods together to make the code easier to read and write. However, you cannot chain methods that return real values, for example, getXXX or isXXX methods.

The following is an example of chaining:

```
btn.setText("Sample").setImage(url).on("click", fun1);
```
Methods for the oraclecrmod Object

This topic describes the methods for the top-level object, oraclecrmod. Table 2 lists the methods associated with events.

Table 2. Methods for oraclecrmod That Are Associated with Events

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Return Type</th>
<th>Description</th>
<th>Sample Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>onReady(cust omFunction)</td>
<td>oraclecrmod</td>
<td>Registers a custom function that is executed by the framework when the DOM tree is constructed. This method is the entry point for running custom code.</td>
<td>oraclecrmod.onReady(function()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>oraclecrmod.createButton({...});</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td>onLoad(cust omFunction)</td>
<td>oraclecrmod</td>
<td>Calls a custom JavaScript function when the onload event is triggered for the document body. This method is similar to onReady(), but it is rarely used because the onReady() method is preferred.</td>
<td>oraclecrmod.onload(onLoadFun);</td>
</tr>
<tr>
<td>onUnload(cust omFunction)</td>
<td>oraclecrmod</td>
<td>Calls a custom JavaScript function when the onunload event is triggered for the document body.</td>
<td>oraclecrmod.onUnload(onUnloadFun);</td>
</tr>
</tbody>
</table>

Table 3 lists the methods that are associated with UI components. The getTitleBar() and getButton() methods use the ID of the title bar or button as a parameter. For information about how to find these IDs, see "Identifying the IDs of Buttons and TitleBars" on page 23.
Table 3. Methods for oraclecrmod That Are Associated with UI Components

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Return Type</th>
<th>Description</th>
<th>Sample Code and Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>getTitleBar(Id)</td>
<td>TitleBar</td>
<td>Returns a TitleBar instance referenced by the Id parameter. You can use this TitleBar instance to add another button to the TitleBar.</td>
<td>var tb = oraclecrmod.getTitleBar(&quot;TitleBarId&quot;);</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>oraclecrmod.createButton({ id:&quot;TestBtn&quot;, text:&quot;Text Button&quot;, parent:tb});</td>
</tr>
<tr>
<td>getButton(Id)</td>
<td>Button</td>
<td>Returns a Button instance referenced by the Id parameter. This method can retrieve both preconfigured buttons and custom buttons.</td>
<td>var btn = oraclecrmod.getButton(&quot;ButtonId&quot;);</td>
</tr>
<tr>
<td>createButton</td>
<td>Button</td>
<td>Creates a custom button. You can call this shortcut method to create a button, instead of using code like the following:</td>
<td>oraclecrmod.createButton({ id:&quot;MyBtnId&quot;, text:&quot;Click to Invoke&quot;, img:&quot;images/yourimage.gif&quot;, disabled:false});</td>
</tr>
<tr>
<td>(config)</td>
<td></td>
<td></td>
<td>Null is returned if a call to createButton() fails. This return value might result, for example, from trying to create a button that has a duplicate ID.</td>
</tr>
</tbody>
</table>
Methods for the TitleBar Object

Table 4 describes the methods that are available for the TitleBar object. The `getButton()` method is useful for the rare cases where preconfigured button IDs are not unique.

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Return Type</th>
<th>Description</th>
<th>Sample Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>getButton</td>
<td>Button</td>
<td>Gets a Button instance with a given button ID within a TitleBar instance.</td>
<td>var tb = oraclecrmod.getTitleBar(&quot;TitleBarID&quot;); var btn = tb.getButton(&quot;ButtonID&quot;);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In most cases, this method works in the same way as the <code>oraclecrmod.getButton()</code> method, and sometimes the <code>oraclecrmod.getButton()</code> method is simpler because you do not have to get a reference to the TitleBar. The only difference is that if there are two buttons that share the same ID, then the <code>oraclecrmod.getButton()</code> method cannot return both buttons. In this case, you can get the TitleBar instance first, then get the Button instance within that TitleBar. Buttons should not have the same ID.</td>
<td></td>
</tr>
</tbody>
</table>

Methods for the Button Object

Table 5 describes the methods that are available for the Button object. For information about defining the event handler for a button, see “Defining an Event Handler for a Button” on page 23. For information about getting the ID of a title bar or button, see “Identifying the IDs of Buttons and TitleBars” on page 23.

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Return Type</th>
<th>Chainable</th>
<th>Description</th>
<th>Sample Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>disable()</td>
<td>this</td>
<td>Yes</td>
<td>Disables the button.</td>
<td>btn.disable();</td>
</tr>
<tr>
<td>enable()</td>
<td>this</td>
<td>Yes</td>
<td>Enables the button.</td>
<td>btn.enable();</td>
</tr>
<tr>
<td>getId()</td>
<td>String</td>
<td>No</td>
<td>Returns the ID of the button. You can use the returned ID to find the button again.</td>
<td>var sBtnId = btn.getId();</td>
</tr>
<tr>
<td>getImage()</td>
<td>String</td>
<td>No</td>
<td>Returns the URL to the image that was assigned to the button.</td>
<td>var sBtnImg = btn.getImage();</td>
</tr>
</tbody>
</table>
### Methods for the Button Object

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Return Type</th>
<th>Chainable</th>
<th>Description</th>
<th>Sample Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>getParent()</td>
<td>TitleBar</td>
<td>No</td>
<td>Returns the parent TitleBar instance of this button. You can call this method to get the parent TitleBar of a known Button and add more buttons to the TitleBar.</td>
<td><code>var tb = btn.getParent(); oraclecrmod.createButton({ id: &quot;NewBtnId&quot;, text: &quot;New Button&quot;, parent: tb });</code></td>
</tr>
<tr>
<td>getText()</td>
<td>String</td>
<td>No</td>
<td>Returns the text of the button.</td>
<td><code>var text = btn.getText()</code></td>
</tr>
<tr>
<td>hide()</td>
<td>this</td>
<td>Yes</td>
<td>Hides the button.</td>
<td><code>btn.hide();</code></td>
</tr>
</tbody>
</table>
| invokeDefault(evt) | Not applicable | No        | Invokes the default event handler for the preconfigured button. This is useful if you want to add additional functionality to a preconfigured button, but then still want to invoke the default event handler. For example, you might want to validate a form before the record is saved. You can redefine the onclick event for the Save button to include your own validation logic. After the validation, you still want to save the record, so you can then call `invokeDefault("click")` on the Save button. | `var btn = oraclecrmod.getButton ("Save Button ID"); btn.offDefault("click").on("click", function(){
  if(data is valid){
    this.invokeDefault ("click");
  }else{
    alert("validation failed");
  }
});` |
| isEnabled()     | Boolean     | No        | Returns whether the button is currently disabled.                                                                                                 | `if(btn.isEnabled()){
  //do something here
}`                                                                                                              |
| isHidden()      | Boolean     | No        | Returns whether the button is hidden.                                                                                                            | `if(btn.isHidden()){
  //do something here
}`                                                                                                              |
### Methods for the Button Object

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Return Type</th>
<th>Chainable</th>
<th>Description</th>
<th>Sample Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>off(evt,custom Function)</td>
<td>this</td>
<td>Yes</td>
<td>Removes the registered event listener for the given event. <strong>NOTE:</strong> The event handler must be the original event handler that was used to register the event. For information about how to define the event handler function, see &quot;Defining an Event Handler for a Button” on page 23.</td>
<td>btn.off(&quot;click&quot;, myClickHandler);</td>
</tr>
<tr>
<td>offAll(evt)</td>
<td>this</td>
<td>Yes</td>
<td>Removes all event handlers attached to the Button for the given event. This method is useful when you want to change the default behavior of a button, for example, to remove the default onclick event.</td>
<td>btn.offAll(&quot;click&quot;);</td>
</tr>
<tr>
<td>offDefault(evt)</td>
<td>this</td>
<td>Yes</td>
<td>Removes the default event handler for the preconfigured button. Because the event handler for preconfigured buttons is not added by the user, you do not have the reference to the original event handler, and so you cannot remove the event handler by calling the off() method.</td>
<td>btn.offDefault(&quot;click&quot;);</td>
</tr>
</tbody>
</table>
### Methods for the Button Object

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Return Type</th>
<th>Chainable</th>
<th>Description</th>
<th>Sample Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>on(evt,customFunction, data)</td>
<td>this</td>
<td>Yes</td>
<td>Registers an event handler for the given event type. The data parameter is an optional parameter. It is useful when you want to pass more information to the customFunction. <strong>NOTE:</strong> You can use an anonymous function as the event handler. However, in that case, you cannot remove the event handler unless you call offAll(), which removes all the registered event handlers for the given event type. So, if you plan to remove the event handler, then do not use the anonymous function as the event handler.</td>
<td>btn.on(&quot;click&quot;, myClickHandler);                                                                                     The following sample code uses the optional data parameter: btn.on(&quot;click&quot;, function(evt, customerNumber) {alert(customerNumber); }, 1234); In this case, the additional data, 1234, is passed to the custom function through the customerNumber parameter. For a further example of using the on() function, see “Defining an Event Handler for a Button” on page 23.</td>
</tr>
<tr>
<td>setImage(url)</td>
<td>this</td>
<td>Yes</td>
<td>Sets an image for the button. You can use a relative URL or absolute URL for an image. If the URL is null, then the image is removed.</td>
<td>btn.setImage(&quot;<a href="http://domain/imgs/img.gif">http://domain/imgs/img.gif</a>&quot;); or btn.setImage(&quot;/images/test.png&quot;);</td>
</tr>
<tr>
<td>setParent(tb)</td>
<td>this</td>
<td>Yes</td>
<td>Sets the given TitleBar as the parent of this button. You can create a button first, without giving it a parent. You can then call this method to add the button to a TitleBar. If a button is already added to a TitleBar, and if you call this method for a different TitleBar, then the calls fails and an alert is displayed.</td>
<td>btn.setParent(titlebar);</td>
</tr>
</tbody>
</table>
Defining an Event Handler for a Button

To define an event handler for a button, you must define a JavaScript function in the following format:

```javascript
function YourFunName(evt, data){
    this.xxxxx();
}
```

The `evt` parameter is the regular event object from the browser. The `data` parameter is the additional data that you want to pass to the event handler. Both of these parameters are optional.

Inside the function body, the `this` keyword points to the current Button object to which this event handler is attached.

You can attach the event handler to your button using this code:

```javascript
btn.on("click",YourFunName,"Your String");
```

In this example, "Your String" is passed as the data parameter to the event handler function `YourFunName()` when the button is clicked.

Identifying the IDs of Buttons and TitleBars

To get a TitleBar or Button instance, you must know the ID of the title bar or button. To find out the ID, you can use the following procedure:

1. Log in to Oracle CRM On Demand in a browser that allows dynamic JavaScript code execution. Examples of such browsers include Mozilla Firefox with Firebug, Google Chrome, and Microsoft Internet Explorer Version 8 and later.
2. Open Developer tools, usually by pressing the F12 button, and go to the Console view.
3. Execute the following JavaScript code:
oraclecrmod.enableIdHint();

4 Make sure that the browser window is the active window.

5 Move your mouse over the title bar or button that you want to get the ID of.
   The browser displays a tooltip near the title bar or button that displays its ID.

## Methods for Getting and Setting Screen Values

This topic describes the methods that you can use to get and set values for fields on the current screen. Table 6 shows the available methods for getting and setting values. The methods are supported for New and Edit pages, but not for Detail pages. For information about finding the field names that you must use when working with these methods, see "Finding Field Names" on page 25.

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Return Type</th>
<th>Description</th>
<th>Sample Code</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>oraclecrmod.getField(field)</td>
<td>A field object exposing the <code>getValue</code> and <code>setValue</code> methods or null, if the field is not found on the screen</td>
<td>Gets the field on the current screen that is identified by the <code>FieldName</code> parameter.</td>
<td><code>oraclecrmod.getField('Parent Account Name')</code></td>
<td>None</td>
</tr>
</tbody>
</table>
Finding Field Names

For methods that reference fields, you must use the correct field names, which you can find from the HTML Field Tag column in the Fields page in the UI for each record type, if your role includes the Upload Client Side Extensions and Manage Custom HTML Head Tag privilege. In recent versions of browsers, you can also use developer tools to find field names.

To find field names in the Fields page

1. In the upper-right corner of any page, click the Admin global link.
2. In the Application Customization section, click the Application Customization link.
3. In the Record Types Setup section, click the link for the required record type.
4. In the Field Management section, click record type Field Setup.

The record type Fields page displays the HTML Field Tag column. This column contains the language-independent field name for each field.

Table 6. Getter and Setter Methods for Screen Values

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Return Type</th>
<th>Description</th>
<th>Sample Code</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>field.getValue()</td>
<td>The field value string</td>
<td>Gets a value for a field in the current screen.</td>
<td><code>oraclecrmod.getField('Location').getValue()</code></td>
<td>For a check box field, the return values are Y for checked, and N for unchecked.</td>
</tr>
<tr>
<td>field.setValue</td>
<td>The new field value string that was just set, or it is null if the method fails</td>
<td>Sets a value for a field in the current screen.</td>
<td><code>oraclecrmod.getField('Type').setValue('Customer')</code></td>
<td>For a check box field, only the values Y or N are accepted for the input parameter. Any other values are ignored. Setting values is not supported for the following: Analytics fields, Currency code fields, Multi-association fields, Single-association fields for which the Auto-Resolve Enabled check box is deselected. Such fields are grayed out on the UI.</td>
</tr>
</tbody>
</table>
To find field names using browser developer tools

1. In New or Edit pages, right click in the field for which the name is required.
2. Select, for example, Inspect element or Inspect with.
   The name of the option varies with browser versions. The browser opens developer tools and highlights the relevant element of the HTML source. The HTML source element has name and id attributes that indicate the field name.
3. Check the field name, which is in the format: FormName.FieldName.
   For example, in AccountEditForm.Location the part after the dot, Location, is the field name.
   In some browser versions you may also see an id attribute with a value like the following, where the part after the final dot is the field name:
   
   A0.R0.Location

   NOTE: For check boxes and multiselect picklist fields, the HTML source element highlighted by developer tools is not the element that indicates the field name. In this case, the next element contains the attributes that indicate the real field name.

Guidelines for Setting Screen Values

When you have multiple fields for which you want to set a value, and if setting a value for one of those fields triggers a page refresh (for example, Sales Stage, Dynamic Layout driving field), then you must call the setValue() method on all the other fields before you call setValue() on the field that triggers the page refresh. Otherwise, all the setValue() calls following the page refresh will be ignored.

Methods for the Context Object

You can get information about the current page and logged-in user to make decisions in your custom JavaScript code, for example, to determine when a script executes.

The methods shown in Table 7 are helper functions that wrap the attributes for the context object shown in Table 8. As a guideline use the helper methods, which are more convenient than using the attributes directly because they return Boolean values. If you use an attribute directly, then you have to do a comparison, such as the following:

   if(oraclecrmod.ctx.pageType == "D")

Whereas using a helper method like the following is simpler and clearer:

   if(oraclecrmod.ctx.isDetailPage())
### Table 7. Helper Methods for the Context Object

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
<th>Parameters and Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>isHomePage</td>
<td>Returns whether the current page type is Homepage</td>
<td>None</td>
</tr>
<tr>
<td>isListPage</td>
<td>Returns whether the current page is a List page</td>
<td>None</td>
</tr>
<tr>
<td>isDetailPage</td>
<td>Returns whether the current page type is Detail page</td>
<td>None</td>
</tr>
<tr>
<td>isEditPage</td>
<td>Returns whether the current page type is Edit page</td>
<td>None</td>
</tr>
<tr>
<td>isNewPage</td>
<td>Returns whether the current page type is New page</td>
<td>None</td>
</tr>
<tr>
<td>isEditOrNewPage</td>
<td>Returns whether the current page type is Edit page or New page</td>
<td>None</td>
</tr>
<tr>
<td>isAdminPage</td>
<td>Returns whether the current page type is Admin page</td>
<td>None</td>
</tr>
<tr>
<td>isObject(object)</td>
<td>Returns whether the current page’s primary object is the record type in the parameter</td>
<td>The String object parameter is the record type, for example: <code>oraclecrmod.ctx.isObject(&quot;Account&quot;)</code></td>
</tr>
</tbody>
</table>

All of the helper methods have a Boolean return type. Table 8 shows the attributes for the context object.

### Table 8. Attributes for the Context Object

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Sample Values and Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>servlet</td>
<td>String</td>
<td>The current servlet path.</td>
<td><code>oraclecrmod.ctx.servlet == &quot;/user/AccountHomePage&quot;</code></td>
</tr>
<tr>
<td>pageType</td>
<td>String</td>
<td>An alphabetic character representing the current page type.</td>
<td>H for Homepage, L for List page, D for Detail page, N for New page, E for Edit page, and A for Admin page.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>if (oraclecrmod.ctx.pageType==&quot;D&quot;) {...}</code></td>
</tr>
<tr>
<td>object</td>
<td>String</td>
<td>The current record type name.</td>
<td>&quot;Account&quot;, &quot;CustomObject4&quot;</td>
</tr>
</tbody>
</table>
Methods for CRUD Operations

This topic describes the methods available for create, read, update and delete (CRUD) operations on off-screen records. Table 9 lists the methods, which are under oraclecrmod.dataSvc. The parameters for the methods are discussed in "Parameters and Return Values for CRUD Methods" on page 30. For a code sample that performs a create operation, see "Code Sample for a Custom Button That Creates a Record" on page 38.

To handle the results of a CRUD operation you must define a callback function as described in "User-Defined Callback Function" on page 32.

TIP: All of the methods run asynchronously, so you must implement the callback function and subsequent function calls to allow for this.

The methods are asynchronous because CRUD operations need to communicate with the server to complete and this involves remote communication across the network using AJAX technology. Because of this, these operations normally take a significantly longer time to complete compared to other operations that happen inside the browser.

### Table 8. Attributes for the Context Object

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Sample Values and Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>layoutId</td>
<td>String</td>
<td>The row ID for the current layout in use, which you can find in the URL of the Page Layout Wizard editing page. For a standard layout, the URL has no row ID, and this attribute has the value of the standard layout name.</td>
<td>&quot;1QA2-P4F90&quot;, &quot;Lead&quot;</td>
</tr>
<tr>
<td>layoutName</td>
<td>String</td>
<td>The display name for the current layout.</td>
<td>&quot;myAcctLayout&quot;</td>
</tr>
<tr>
<td>roleId</td>
<td>String</td>
<td>The row ID for the current user's role, which you can find in the URL of the Role Management Wizard editing page.</td>
<td>&quot;1-G4WZ0&quot;</td>
</tr>
<tr>
<td>roleName</td>
<td>String</td>
<td>The name of the current user role, unlocalized</td>
<td>&quot;Administrator&quot;</td>
</tr>
<tr>
<td>lang</td>
<td>String</td>
<td>The current user's language as a three-character code</td>
<td>&quot;enu&quot;, &quot;deu&quot;</td>
</tr>
</tbody>
</table>

### Methods for CRUD Operations

This topic describes the methods available for create, read, update and delete (CRUD) operations on off-screen records. Table 9 lists the methods, which are under oraclecrmod.dataSvc. The parameters for the methods are discussed in "Parameters and Return Values for CRUD Methods" on page 30. For a code sample that performs a create operation, see "Code Sample for a Custom Button That Creates a Record" on page 38.

To handle the results of a CRUD operation you must define a callback function as described in "User-Defined Callback Function" on page 32.

TIP: All of the methods run asynchronously, so you must implement the callback function and subsequent function calls to allow for this.

The methods are asynchronous because CRUD operations need to communicate with the server to complete and this involves remote communication across the network using AJAX technology. Because of this, these operations normally take a significantly longer time to complete compared to other operations that happen inside the browser.
In the browser, all the JavaScript code runs in a single thread. If synchronous calls were used for the CRUD operations, the browser would not be able to handle any user interaction when the CRUD operations were in process. Asynchronous calls are therefore used for CRUD operations. During the CRUD operations the user can still interact with the page, and when the operation is finished, the framework calls the callback function to handle the result. This results in a much better user experience.

Table 9. Create, Read, Update, and Delete Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Return Type</th>
<th>Description</th>
<th>Sample Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>createRecord</td>
<td>JSONObject</td>
<td>Creates an off-screen record. The parameters specify the record type, and field name-value pairs for the record. The method returns the row ID and mod ID of the record.</td>
<td><code>oraclecrmod.dataSvc.createRecord(&quot;Account&quot;, {&quot;Name&quot;: &quot;Account 1&quot;, &quot;Account Contact Role&quot;: &quot;Admin&quot;, &quot;Type&quot;: &quot;Customer&quot;, &quot;Description&quot;: &quot;Description Test&quot;}, null, callback);</code></td>
</tr>
<tr>
<td>readRecord</td>
<td>JSONObject</td>
<td>Retrieves fields not exposed for the current record or from an off-screen record. The parameters specify the record type, list of field names, and search type and search specification. The method returns a list of the field names and their values.</td>
<td><code>oraclecrmod.dataSvc.readRecord(&quot;Account&quot;, {&quot;Name&quot;: &quot;Description&quot;, { searchType: &quot;rowId&quot;, rowId&quot;: &quot;1QA2-TNRWW&quot;}}, callback);</code></td>
</tr>
</tbody>
</table>
For create and update operations on required and read-only fields in off-screen records, only default preconfigured fields and not custom fields are supported.

If a preconfigured required field is missing, then the create or update operation fails. If an operation attempts to populate a read-only field, then it is ignored.

### Parameters and Return Values for CRUD Methods

The following are the parameter values for the CRUD methods listed in Table 9:

- **objectName**. The name of the record type, for example, "Account"
- **fieldNameNames (for readRecord)**. A list of business component field names for which values are returned. You can specify the list of fields as a comma-separated string, or as an array. For example, the string:

  "Name, Location, Type, Description"

  or the array:

  ["Name", "Location", "Type", "Description"]

---

### Table 9. Create, Read, Update, and Delete Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Return Type</th>
<th>Description</th>
<th>Sample Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>updateRecord (objectName, fieldNameValuePairs , updateParameters, callback)</td>
<td>JSONObject</td>
<td>Updates an off-screen record. The required parameters specify the record type, field name-value pairs, and search type and search specification. The method returns the row ID and mod ID of the updated record.</td>
<td>oraclecrmod.dataSvc.updateRecord(&quot;Account&quot;, {&quot;Name&quot; : &quot;Account 1&quot;, &quot;Account Contact Role&quot; : &quot;Admin&quot;, &quot;Type&quot; : &quot;Customer&quot;, &quot;Description&quot; : &quot;Description Test&quot;}, {searchType: &quot;rowId&quot;, &quot;rowId&quot; : &quot;1QA2-TNRWW&quot;}, callback);</td>
</tr>
<tr>
<td>deleteRecord (objectName, deleteParameters, callback)</td>
<td>JSONObject</td>
<td>Deletes an off-screen record. The required parameters specify the record type, and search type and search specification. The method deletes the specified record and returns the deletion results.</td>
<td>oraclecrmod.dataSvc.deleteRecord(&quot;Account&quot;, &quot;1QA2-TNRWW&quot;, callback);</td>
</tr>
</tbody>
</table>
**fieldNameValuePairs (for createRecord and updateRecord).** An object containing field name-value pairs. For example:

```javascript
{ "Name": "Account 1", "Account Contact Role": "Admin", "Type": "Customer", "Description": "Description Test" }
```

For create operations, if a required field is missing, then the operation fails. Read-only fields are ignored. For update operations, if a required field is missing a value, then the operation fails. Attempts to populate read-only fields are ignored.

**readParameters, updateParameters, deleteParameters.** A search type and search specification. For example:

```javascript
{ searchType: rowId, rowId: "1QA2-TNRWW" }
```

You can also specify just a string, which by default means search by row ID, for example:

```
"1QA2-TNRWW"
```

**createParameters.** A value for a parameter reserved for future use. Specify the value null.

**callback.** A user-defined function to process the results of the CRUD operations. For more information, see “User-Defined Callback Function” on page 32.

All of the CRUD methods return a JSONObject object, which has the following properties:

- **status.** The status as a string: 'OK' or "ERROR".
- **errors.** Any errors in a JSONArray object. The format of the error object returned is as follows:

  ```javascript
  [Object {Code="MSG_NUM", Message="MSG_TEXT"}]
  ```

- **fieldNameValuePairs.** A list of the field names and values in a JSONArray object. The format of the object is as follows or null:

  ```javascript
  [Object {Name="Account Val", Location="Location Val", Type="Partner", more...}]
  ```

**Finding Record Type Names**

For methods like the CRUD methods that reference record types, you must use the correct record type names.

To find record type names, you can:

- Use the oraclecrmod.ctx.isObject() method, see “Methods for the Context Object” on page 26.
- Find the name in the URL for the record type Application Customization page in the UI. The value for the ActiveObj parameter in the URL gives the record type name.

**Guidelines for Performing CRUD Operations**

For update operations on records, first do a read operation and obtain an object, then change the object values, and perform the update operation.
User-Defined Callback Function

You can use a callback function to handle responses to CRUD operations, as shown in Table 10.

Table 10. Callback Function for CRUD Operations

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
<th>Sample Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>callback (request, response)</code></td>
<td>User-defined JavaScript function to process the results of the CRUD operations.</td>
<td><code>callback(request, response) { }</code></td>
</tr>
</tbody>
</table>

The parameters for a callback function are as follows:

- **request.** The request object containing the command, row ID, and other field objects passed in the initial request.
- **response.** The response object containing the result of the CRUD operation. The object properties are as follows:
  - **status.** The status of the request, which is either OK or ERROR.
  - **data.** The name-value pair of the fields whose values are returned (fieldNameValuePairs). For example:
    ```javascript
    Name="Account Test", Location="Account Location", Type="Competitor"
    ```
  - **errors.** Any error returned, which can be null or an error object.
- **helper function.** The name of a helper function, for example, `getRowId()`, `getModId()`. For more information, see “Helper Functions for Callback Functions” on page 32.

Helper Functions for Callback Functions

Table 11 describes the helper functions that are available for callback functions.

Table 11. Helper Functions for Callback Functions

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Return Type</th>
<th>CRUD Operations</th>
<th>Description</th>
<th>Sample Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getRowId()</code></td>
<td>row ID</td>
<td>Create, read, update</td>
<td>Gets the row ID of the record in the response object.</td>
<td><code>var rowID = response.getRowId();</code></td>
</tr>
<tr>
<td><code>getModId()</code></td>
<td>mod ID</td>
<td>Create, Read, Update</td>
<td>Gets the mod ID of the record in the response object.</td>
<td><code>var modId = response.getModId();</code></td>
</tr>
<tr>
<td><code>getFieldValue</code></td>
<td>Field value</td>
<td>Read</td>
<td>Gets the field value of the corresponding field name passed in the function.</td>
<td><code>var desc = response.getFieldValue(&quot;Description&quot;);</code></td>
</tr>
</tbody>
</table>
Sample Code for Callback Function

The following is an example of code for a callback function:

```javascript
function callback(request, response){
    if (response.status == "OK")
    {
        //Using helper functions to get value
        var desc = response.getFieldValue("Description");
        var type = response.getFieldValue("Type");
        var rowID = response.getRowId();
    }
}
```

Refer to "Example Code of How to Deal with the Error Object" on page 33 for an example of a callback function that performs error handling.

Errors and Error Handling

For CRUD operations, the errors object specifies null or an array of objects in the following format:

```javascript
{
    fac : "SBL-ODU-MSG_NUMBER",   // error code
    msg : "MSG_TEXT"   // error message
}
```

For example, for response.errors[0]:

```javascript
{
    fac : "SBL-ODU-00271",
    msg : "Invalid value for the parameter:objectName."
}
```

Example Code of How to Deal with the Error Object

You must implement your own error handling in custom JavaScript code. As an example of how to deal with the error object, the following is a callback function that displays error codes and error messages from CRUD operations.

```javascript
function callback(request, response){
    var data = "data: ";
    var status = "status: " + response.status;
```
```javascript
var error = "error: 

if (status == "OK")
    error += status;
else
    error += response.errors[0].fac + '\n' + response.errors[0].msg;

var dataObj = response.data;
if (dataObj != null)
    data += "Id = " + response.getRowId() +", Mod Id = " + response.getModId();
else
    data += "Data is Null";

alert(data + '\n' + status + '\n' + error);
```
### Table 12. Error Codes and Messages

<table>
<thead>
<tr>
<th>Code</th>
<th>Message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBL-DAT-00398</td>
<td>Field 'field_name' does not exist in definition for business component 'record_type'. Please ask your systems administrator to check your configuration.</td>
<td>An operation specified a nonexistent field name.</td>
</tr>
<tr>
<td>SBL-DAT-00498</td>
<td>'&lt;field&gt;field_name&lt;/field&gt;' is a required field. Please enter a value for the field.</td>
<td>An operation did not specify a value for a field that is required.</td>
</tr>
<tr>
<td>SBL-ODS-50006</td>
<td>A record that contains identical values to the record you are trying to create already exists. If you would like to enter a new record, please ensure that the field values are unique.</td>
<td>An operation tried to create a record that already exists.</td>
</tr>
</tbody>
</table>
This appendix contains code samples for the public JavaScript APIs available with Oracle CRM On Demand.

**NOTE:** When including JavaScript code within HTML, remember to include the code within `<script>` tags.

This appendix includes the following topics:

- Code Sample for Creating a Custom Button for Validation
- Code Sample for a Custom Button That Creates a Record on page 38
- Code Sample for a Custom Button That Updates a Record on page 39
- Code Sample for a Custom Button That Gets a Shipping Address to Pass to an External Site on page 41
- Code Sample for Changing the Behavior of a Save Button on page 42

### Code Sample for Creating a Custom Button for Validation

The following sample code creates a custom button labeled *Validate* on the Opportunity Detail Page. When the button is clicked, a validate function is called to validate the values of the Primary Revenue Amount and Next Step fields. The sample code also hides the Add button on the Contact related information applet.

```javascript
// entry point for running custom code
oraclecrmod.onReady(function()
{
  // when on the Opportunity Detail page
  if(oraclecrmod.ctx.object == "Opportunity" && oraclecrmod.ctx.isDetailPage())
  {
    // define validate function
    function validate()
    {
      var revenue = oraclecrmod.getField("Primary Revenue Amount").getValue();
      var next step = oraclecrmod.getField("Next Step").getValue();
      // validate custom business logic goes here based on field values retrieved
    }
  }

// hide Add button
```

// entry point for running custom code
oraclecrmod.onReady(function()
{
  // when on the Opportunity Detail page
  if(oraclecrmod.ctx.object == "Opportunity" && oraclecrmod.ctx.isDetailPage())
  {
    // define validate function
    function validate()
    {
      var revenue = oraclecrmod.getField("Primary Revenue Amount").getValue();
      var next step = oraclecrmod.getField("Next Step").getValue();
      // validate custom business logic goes here based on field values retrieved
    }
  }

// hide Add button
```
// get the title bar
titleBar = oraclecrmod.getTitleBar("OpportunityFormTB");
// create the new Validate button
button = oraclecrmod.createButton({id:"ValidateButton", text:"Validate", parent:titleBar});
// associate the validate function with the button click event
button.on("click", validate);
// get the Add button and hide it
oraclecrmod.getButton("BTN_TB_ContactRoleChildList_ContactRoleNewNav").hide();
}
});

Code Sample for a Custom Button That Creates a Record

The following code sample illustrates the use of the createRecord() method to create an Account record with three fields, Location, Type, and Description, when a button created with the createButton() method and labeled Test Create is clicked.

oraclecrmod.onReady(
    function()
    {
        if(oraclecrmod.ctx.isObject("Account") && oraclecrmod.ctx.isDetailPage())
        {
            var callback = function(request,response)
            {
                var data = "Response Data: ";
                var status = response.status;
                var error = "Error message: ";
                if (status == "OK")
                {
                    error += status;
                }
                else
                    error += status;
            }
            ...
```javascript
{  
    error += response.errors;  
}
var dataObj = response.data;
if (dataObj != null)
{
    data += "Id = " + response.getRowId() + ", Mod Id = " + response.getModId();
} else
{
    data += "Data is Null";
}
alert(data + "\n" + status + "\n" + error);
}
var createRecord = function()
{
    oraclecrmod.dataSvc.createRecord("Account", {Name : "Account Name Create123", Location : "Location Test", Type : 'Customer', Description : "Description Test"}, "", callback);
};
var tb = oraclecrmod.getTitleBar("AccountFormTB");
var bt = oraclecrmod.createButton({id:"TestCreateBtn", text:"Test Create", parent:tb});
bt.on("click",createRecord);
}
});

Code Sample for a Custom Button That Updates a Record

The following code sample illustrates the use of the updateRecord() method to update an Account record when a button created with the createButton() method and labeled Test Update is clicked.
oraclecrmod.onReady(
```
```javascript
function() {
    if(oraclecrmod.ctx.isObject("Account") && oraclecrmod.ctx.isDetailPage()) {
        var callback = function(request, response) {
            var data = "Response Data: ";
            var status = response.status;
            var error = "Error message: ";
            if (status == "OK") {
                error += status;
            } else {
                error += response.errors;
            }
            var dataObj = response.data;
            if (dataObj != null) {
                data += "Id = " + response.getRowId() + ", Mod Id = " + response.getModId();
            } else {
                data += "Data is Null";
            }
            alert(data + "\n" + status + "\n" + error);
        };
        var updateRecord = function() {
            oraclecrmod.dataSvc.updateRecord("Account", {Name : "Account Name Update123"}, "AUDA-1HSXSB", callback);
        };
    }
}
```
var tb = oraclecrmod.getTitleBar('AccountFormTB');
var bt = oraclecrmod.createButton({id: "TestUpdateBtn", text: "Test Update", parent: tb});
bt.on("click", updateRecord);
});

### Code Sample for a Custom Button That Gets a Shipping Address to Pass to an External Site

The following code creates a custom button labeled **Map Shipping Address** on the Account Detail Page. When the button is clicked, the shipping address is passed to an external URL, in this case, the URL for Google Maps.

```javascript
// example: create a map button
// entry point for running custom code
oraclecrmod.onReady(function()
{
  // when on the Account Detail page
  if(oraclecrmod.ctx.object == "Account" && oraclecrmod.ctx.isDetailPage())
  {
    // define map function
    function map()
    {
      var wholeAddress = "";
      if (oraclecrmod.getField("Ship To Street Address") != null)
        wholeAddress += oraclecrmod.getField("Ship To Street Address").getValue() + ", ";
      if (oraclecrmod.getField("Ship Street Address 2") != null)
        wholeAddress += oraclecrmod.getField("Ship Street Address 2").getValue() + ", ";
      if (oraclecrmod.getField("Ship Street Address 3") != null)
        wholeAddress += oraclecrmod.getField("Ship Street Address 3").getValue() + ", ";
      if (oraclecrmod.getField("Ship To County") != null)
```
```javascript
wholeAddress += oraclecrmod.getField("Ship To County").getValue() + " ";
if (oraclecrmod.getField("Ship To Postal Code") != null)
wholeAddress += oraclecrmod.getField("Ship To Postal Code").getValue() + " ";
if (oraclecrmod.getField("Ship To Country") != null)
wholeAddress += oraclecrmod.getField("Ship To Country").getValue() + " ";

window.open("http://maps.google.com?q=" + encodeURIComponent(wholeAddress));

// get the title bar
titleBar = oraclecrmod.getTitleBar("AccountFormTB");
// create the new map button
button = oraclecrmod.createButton({id:"mapButton", text:"Map Shipping Address", parent:titleBar});
// associate the map function with the button click event
button.on("click", map);
```
});

});

});
JavaScript API Code Samples ■ Code Sample for Changing the Behavior of a Save Button
Index

B
Button class 16
Button object
  event handler 23
  IDs 23
  methods 19
buttons
  changing behavior 8
  contexts for customization 9
  creating 7
  disabling 8
  event handler 23
  hiding 8
  look and feel 8

callback function 32
classes
  Button 16
  Context 16
  TitleBar 15
code sample
  callback function 33
  changing behavior of a button 42
  create operation 38
Context class 16
Context object
  attributes 27
  guidelines 26
  methods 26
createButton method 18
createRecord method 29
CRUD operations
  guidelines 31
  methods 28
  parameters 30
  return values 30
custom buttons
  contexts for customization 9
  creating 7
custom HTML head additions 13
delteRecord method 30
disable method 19
disableCustomJS URL parameter 13

e
enable method 19
errors
  codes 34
  handling 33
  messages 34
  object 32, 33

F
field names 25

G
getButton method 18, 19
getField method 24
getFieldValue helper function 32
getId methods 19
getAddress method 19
getModId helper function 32
getParent method 20
getAddress helper function 32
getter and setter methods 24
getText method 20
getAddress method 18
getAddress method 25

H
helper functions
  getFieldId 32
getAddress 32
getAddress 32
hide method 20
HTML head additions 13

I
invokeDefault method 20
isAdminPage method 27
isDetailPage method 27
isDisabled method 20
isEditOrNewPage method 27
isEditPage method 27
isHidden method 20
isHomePage method 27
isListPage method 27
isNewPage method 27
isObject method 27, 31

J
JavaScript libraries 11

M
methods
Button object 19
chaining of 16
Context object 26
createButton 18
createRecord 29
CRUD operations 28
deleteRecord 30
disable 19
enable 19
getButton 18, 19
getField 24
getId 19
getImage 19
getParent 20
getter and setter methods 24
getText 20
getTitleBar 18
getValue 25
hide 20
invokeDefault 20
isAdminPage 27
isDetailPage 27
isDisabled 20
isEditOrNewPage 27
isEditPage 27
isHidden 20
isHomePage 27
isListPage 27
isNewPage 27
isObject 27, 31
off 21
offAll 21
offDefault 21
on 22
onLoad 17
onReady 17
onUnload 17
oraclecrmod object 17
readRecord 29
setImage 22
setParent 22
setText 23
setValue 25
show 23
TitleBar object 19
updateRecord 30

O
off method 21
offAll method 21
offDefault method 21
on method 22
onLoad method 17
onReady method 17
onUnload method 17
oraclecrmod object
methods 17
namespace 15

P
privileges 11

R
readRecord method 29
record type names 31

S
screen values
getting 24
guidelines for setting 26
setting 24
security 14
setImage method 22
setParent method 22
setText method 23
setValue method 25
show method 23

T
TitleBar class 15
TitleBar object
IDs 23
methods 19

U
updateRecord method 30