# Oracle CRM On Demand

## JavaScript API Developer's Guide Release 41

August 2020

Oracle CRM On Demand
JavaScript API Developer's Guide Release 41

August 2020

Part Number: F30072-01



Authors: Oracle CRM On Demand Information Development Team

# Contents

# Preface

This preface introduces information sources that can help you use the application and this guide.

## Using Oracle Applications

To find guides for Oracle Applications, go to the Oracle Help Center at *http://docs.oracle.com/*.

## Contacting Oracle

### Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit *My Oracle Support* or visit *Accessible Oracle Support* if you are hearing impaired.

### Comments and Suggestions

Please give us feedback about Oracle Applications Help and guides! You can send an e-mail to: *oracle_fusion_applications_help_ww_grp@oracle.com*.

# 1 What's New in This Release

## What's New in Oracle CRM On Demand JavaScript API Developer's Guide, Release 41

No new features have been added to this guide for this release. This guide has been updated to reflect only product version changes.

# 2 Overview of the JavaScript API

## 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*
- *Color Coding of Fields and Rows*

**Note:** Custom JavaScript code and other customized code can be disabled and enabled in Oracle CRM On Demand pages. For more information, see *About Enabling and Disabling Customized Code in Oracle CRM On Demand*.

## 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
- To add tooltips for 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*.

### 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*.

# 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*.

# Creating Tooltips for Buttons

The JavaScript API allows you to specify tooltip text when you create a custom button and provides a method to specify tooltip text for multiple existing buttons in a single call.

You can use methods to get and set tooltip text for a button and to get and set the tooltip format, which can be HTML or plain text. For information about these methods, see *Methods for the Button Object*.

# 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*.

# 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* and *Defining an Event Handler for a Button*.

### 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 all fields, language dependent code (LDC) values are retrieved, however for picklists, there are also additional getter and setter methods to retrieve and set language independent code (LIC) values. For information about these methods, see *Methods for the Field Object* and *Getting and Setting LIC Values of Picklists*.

### 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.
- **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*.

**Note:** For the CRUD operations, only top-level record types and not child-level record types are supported.

**Tip:** Use the Oracle CRM On Demand REST API for data manipulation with off-screen records. The REST API is more powerful for data manipulation of off-screen records and it has more functionality such as support for CRUD operations on child-level record types. For more information, see *Oracle CRM On Demand REST API Developer's Guide* .

## 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*.

# 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*.

# Color Coding of Fields and Rows

You can use methods of the JavaScript API to highlight fields and rows in different colors in the Oracle CRM On Demand UI when particular conditions are met. You can change the colors of the following:

- In Detail pages, the background color and text color of field labels and field values.
- In Edit pages, the background color and text color of field labels and the background color of field values.
- In List pages (and related item lists), the background color and text color of field values. Also, the background color for entire rows.

As an example, on account records the Account Name field might be used for highlighting key accounts. On the Account Detail page the color of the Account Name field could change, for example, to green, when the sales representative changes the value of the Account Revenues field to a threshold value. On the Account List page, rows for key account records would be highlighted in green.

For a code sample, see *Code Sample for Color Coding of Fields and Rows*.

The color coding that is set using the JavaScript API takes precedence over system or theme colors. For example, if you set a required field to another text color, the color coding overrides the red text color for the required field.

# Color Coding of Fields in Detail and Edit Pages

To support color coding in Detail and Edit pages, a Form object is required and a custom handler must be registered for the form. For information about methods of the Form object, see the table in *JavaScript API Reference*, and for an example of a custom handler, see *Example of a Custom Handler for a Form Object*.

The custom handler is automatically executed when the form initially loads or refreshes due to any other actions on the screen such as inline editing. In the custom handler you can use methods of the Field object to get and set colors for the labels and values of fields, see the table in *JavaScript API Reference*.

# Color Coding of Fields and Rows in Lists

To support color coding in lists (on List pages and in related item lists), a List object and ListRow object are required respectively for the list, and specific rows in the list. A custom handler must be registered for the List object and this handler is automatically executed on initial page load, on paging, and when a field on any of the rows is edited inline. For information about methods of the List object, see the table in *Methods for the List Object*, and for an example of a custom handler, see *About the Custom Handler for a List Object*.

In the custom handler you can use:

- Methods of the ListRow object to get and set the background color of rows, see the table in *Methods for the ListRow Object*.
- Methods of the Field object to get and set colors for the values of fields within rows, see the table in *JavaScript API Reference*.

**Note:** In lists, if both the background and text color for a specific field within a row, and the background color for the entire row are set using the JavaScript API, then within the row, the color coding for the specific field overrides the color coding for the row.

For information about finding the field names that you must use when working with these methods, see *Finding Field Names*.

For information about finding list names, see *Finding List Names*.

# Restrictions That Apply to Color Coding of Fields

Using the JavaScript API, you can set the color coding of entire concatenated fields. However, the color coding of constituent fields of a concatenated field is ignored, as partial coloring of a field is not possible. For example, you might have the Account Name field included in a concatenated field. The field value text color of Account Name may be coded to red, but the Account Name part of the concatenated field will not be red unless the field value text color for the whole concatenated field is coded as red.

There is no display text for check boxes, or for visual indicator fields such as Star Rating and Stoplight fields, therefore attempts to set text colors on these fields are ignored.

# 3 Getting Started with the JavaScript API

## Getting Started with the JavaScript API

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*
- *Managing HTML Head Additions*
- *Guidelines for Managing HTML Head Additions*
- *About Enabling and Disabling Customized Code in Oracle CRM On Demand*
- *Security Considerations*

## 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*.

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*.

## 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:

     ```
     <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:

   ```
   <script type="text/javascript" src="../user/content/Test.js"></script>
   ```

   The following is a reference to an external file:

   ```
   <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.

# About Enabling and Disabling Customized Code in Oracle CRM On Demand

Administrators can enable and disable custom JavaScript code and other types of customized code for a user by setting the Customized Code Enablement field in a user's profile. Also, if the administrator adds the Customized Code Enablement field to the User Owner page layout for a user role, then users who have that role can disable and enable customized code for themselves by setting the Customized Code Enablement field in their personal profile.

Disabling all customized code can be helpful in troubleshooting technical issues you experience while working in Oracle CRM On Demand. If the issue no longer occurs when the customized code is disabled, then it is likely that the customized code is the cause of the issue.

The Customized Code Enablement picklist field determines whether all customized code on the pages in Oracle CRM On Demand is enabled or disabled for the user, and whether the customized code indicator is enabled or disabled for the user. The following options are available:

- **Enabled**. This is the default setting in the standard application. When this option is selected, all of the customized code that is available in the pages in Oracle CRM On Demand is enabled, but the customized code indicator is not enabled.
- **Enabled with indicator**. When this option is selected, all of the customized code that is available in the pages in Oracle CRM On Demand is enabled. In addition, the customized code indicator is enabled.
- **Disabled with indicator**. When this option is selected, all of the customized code that is available in the pages in Oracle CRM On Demand is disabled. In addition, the customized code indicator is enabled.

When the customized code indicator is enabled, one of the following messages appears at the bottom of each page that you access in Oracle CRM On Demand:

- **Customized code active**. Customized code is detected and is active in the current page.
- **Customized code not detected**. No customized code is detected in the current page.
- **Customized code disabled**. Customized code is detected in the current page, but the customized code is disabled.

When customized code is active in a page, the customized code is visible in the source code for the page, with comments to mark the start and end of the customized code. When the customized code is disabled, the customized code is not included in the source code for the page. Instead, the source code includes a comment to indicate that the customized code is disabled.

For more information about enabling and disabling customized code see *Oracle CRM On Demand Online Help* .

# 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.

# 4 JavaScript API Reference

## JavaScript API Reference

This chapter describes the public JavaScript application programming interfaces (API) available with Oracle CRM On Demand. It includes the following topics.



## 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 the table in *Identifying the IDs of Buttons and TitleBars*.

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 the second table in *Methods for the oraclecrmod Object* 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 the second table in *Methods for the oraclecrmod Object* and the table in *Methods for the TitleBar Object* for information about the API methods used to get Button IDs and create buttons, and the table in *Methods for the Button Object* 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*, in particular, the second table in the topic.

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 the first table in *Methods for the Context Object*.

# 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. The following information lists the methods associated with events.

| Method Name | Return Type | Description | Sample Code |
|---|---|---|---|
| onReady(customFunction | oraclecrmod | 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. | `oraclecrmod.onReady(function() { oraclecrmod.createButton({...}); });` |
| onLoad(custom Function) | oraclecrmod | Calls a custom JavaScript function when the onload event is triggered for the document body.<br><br>This method is similar to onReady(), but it is rarely used because the onReady() method is preferred. | `oraclecrmod.onLoad(onLoadFun);` |
| onUnload(customFunctio | oraclecrmod | Calls a custom JavaScript function when the onunload event is triggered for the document body. | `oraclecrmod.onUnload(onUnloadFun);` |

The following table 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*.

| Method Name | Return Type | Description | Sample Code and Notes |
|---|---|---|---|
| getTitleBar(Id) | TitleBar | Returns a TitleBar instance referenced by the Id parameter. You can use this TitleBar instance to add another button to the TitleBar. | `var tb = oraclecrmod.getTitleBar("TitleBarId"); oraclecrmod.createButton( {id:"TestBtn",text:"Text Button", parent:tb});` |
| getButton(Id) | Button | Returns a Button instance referenced by the Id parameter. | `var btn = oraclecrmod.getButton("ButtonId");` |

| Method Name | Return Type | Description | Sample Code and Notes |
|---|---|---|---|
| | | This method can retrieve both preconfigured buttons and custom buttons. | |
| createButton (config) | Button | Creates a custom button. You can call this shortcut method to create a button, instead of using code like the following:<br>`new oraclecrmod.component.Button(config);`<br>The config parameter has the following properties:<br>• **id**. The ID of the button (String).<br>• **text**. The display text of the button (String).<br>• **img**. The URL of the image used by the button (String).<br>• **disabled**. Whether the button is disabled (Boolean).<br>• **display**. Whether the button is displayed (Boolean).<br>• **toolTipText**. The text to be displayed in the tooltip of the button (String).<br>• **toolTipFormat**. The format of the tooltip text to be displayed (HTML or Plain Text, which is the default). | `oraclecrmod.createButton({ id:"MyBtnId", text:"Click to Invoke", img:"images/yourimage.gif", disabled:false, toolTipText:"<p>Click this</p>", toolTipFormat:"HTML"});`<br>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.<br>For more information about tooltip formatting, see *Considerations for Specifying Tooltips with HTML Formatting*. |
| registerButtonToolTi (tooltipInfoList) | | Registers tooltips for a set of buttons in one call.<br>The input object is an array of button names and the associated tooltip text and tooltip format for each button.<br>For more information about tooltip formatting, see *Considerations for Specifying Tooltips with HTML Formatting*. | `var tooltipInfoList = [ ["BTN_TB_AccountForm_AccountNewNav", "<p>Helps to create new account</p>", "HTML"], ["BTN_TB_AccountForm_AccountEditNav"," Helps to edit the existing account", "Plain Text"], ["BTN_TB_AccountForm_AccountPreCopyNav ", "Helps to copy this account and create new one", "Plain Text"] ]; oraclecrmod.registerButtonToolTip(tool tipInfoList);` |
| getList(listname) | List | Gets a List object on a List or Detail page, on which a custom display handler can be registered or unregistered. | `listObj = oraclecrmod.getList("AccountList"); listObj = oraclecrmod.getList("AddressChildList"); listObj = oraclecrmod.getList();` |

| Method Name | Return Type | Description | Sample Code and Notes |
|---|---|---|---|
| | | The List object is the starting point for getting Row objects from the list, and Field objects from rows. For more information, see *Methods for the List Object*. | If there are no lists with the listname value on the screen, null is returned In full list pages there is only one list, so the listname parameter can be omitted. For information about finding list names, see *Finding List Names*. |
| getForm() | Form | Gets a Form object on an Edit or Detail page, on which a custom display handler can be registered or unregistered.<br><br>For more information, see *Methods for the Form Object*. | `oraclecrmod.getForm();`<br><br>If there is no Form object on the screen, then null is returned.<br><br>If getForm() is used, for example, on a homepage or list page, null is returned. |

# Considerations for Specifying Tooltips with HTML Formatting

When calling the createButton(), registerButtonToolTip(), or setToolTipText() methods, you can specify tooltip text with HTML formatting. When specifying tooltip text with HTML formatting, performance is better if you avoid complex HTML and reduce the number of images displayed in the tooltip. Interactive controls, such as hyperlinks, are not supported. It is also recommended that you specify the padding properties of elements to make the tooltip text easier to read.

Quote characters in HTML code must be escaped, otherwise the entire script does not render on the page at all, and the tooltip does not appear. The following is an example of HTML code with padding specified and with the quotes escaped:

```
<div style=\"background-color:red;padding:3px 8px;border-radius:5px;\">This is a div</div>
```

**Note:** When you use the HTML tag <DIV>, it is recommended to use a border to ensure that the content in the <DIV> tag stays in the tooltip box.

# Methods for the TitleBar Object

The following 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.

| Method Name | Return Type | Description | Sample Code |
|---|---|---|---|
| getButton (buttonId) | Button | Gets a Button instance with a given button ID within a TitleBar instance.<br><br>In most cases, this method works in the same way as the oraclecrmod.getButton() method, and sometimes the oraclecrmod.getButton() 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 oraclecrmod.getButton() method cannot return both buttons. In this case, you can get the TitleBar | `var tb = oraclecrmod.getTitleBar ("TitleBarID");`<br><br>`var btn = tb.getButton("ButtonID");` |

| Method Name | Return Type | Description | Sample Code |
|---|---|---|---|
| | | instance first, then get the Button instance within that TitleBar. Buttons should not have the same ID. | |

# Methods for the Button Object

The following 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*. For information about getting the ID of a title bar or button, see *Identifying the IDs of Buttons and TitleBars*.

| Method Name | Return Type | Chain-able | Description | Sample Code |
|---|---|---|---|---|
| disable() | this | Yes | Disables the button. | `btn.disable();` |
| enable() | this | Yes | Enables the button. | `btn.enable();` |
| getId() | String | No | Returns the ID of the button. You can use the returned ID to find the button again. | `var sBtnId = btn.getId();` |
| getImage() | String | No | Returns the URL to the image that was assigned to the button. | `var sBtnImg = btn.getImage();` |
| getParent() | TitleBar | No | 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. | `var tb = btn.getParent();`<br>`oraclecrmod.createButton({ id: "NewBtnId",text: "New Button", parent:tb});` |
| getText() | String | No | Returns the text of the button. | `var text = btn.getText();` |
| getToolTipText() | String | No | Returns the tooltip text of the button. | `var toolTipText = btn.getToolTipText();` |
| getToolTipFormat() | String | No | Returns the tooltip format of the button | `var toolTipFormat = btn.getToolTipFormat();` |
| hide() | this | Yes | Hides the button. | `btn.hide();` |
| invokeDefault (evt) | Not applic-able | 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 | `var btn = oraclecrmod.getButton ("Save Button ID");`<br>`btn.offDefault("click").on("click", function()`<br>`{if(data is valid){` |

| Method Name | Return Type | Chain-able | Description | Sample Code |
|---|---|---|---|---|
| | | | still want to invoke the default event handler.<br><br>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. | `this.invokeDefault("click");`<br>`}else{`<br>`alert("validation failed");`<br>`}`<br>`});` |
| isDisabled() | Boolean | No | Returns whether the button is currently disabled. | `if(btn.isDisabled()){`<br>`//do something here`<br>`}` |
| isHidden() | Boolean | No | Returns whether the button is hidden. | `if(btn.isHidden()){`<br>`//do something here`<br>`}` |
| off(evt, customFunction) | this | Yes | Removes the registered event listener for the given event.<br><br>**Note:** The event handler must be the original event handler that was used to register the event.<br><br>For information about how to define the event handler function, see *Defining an Event Handler for a Button*. | `btn.off("click",myClickHandler);` |
| offAll(evt) | this | Yes | Removes all event handlers attached to the Button for the given event.<br><br>This method is useful when you want to change the default behavior of a button, for example, to remove the default onclick event. | `btn.offAll("click");` |
| offDefault(evt) | this | Yes | 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. | `btn.offDefault("click");` |

| Method Name | Return Type | Chain-able | Description | Sample Code |
|---|---|---|---|---|
| on(evt, customFunction, data) | this | Yes | Registers an event handler for the given event type.<br><br>The data parameter is an optional parameter. It is useful when you want to pass more information to the customFunction.<br><br>**Note:** 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. | `btn.on("click",myClickHandler);`<br><br>The following sample code uses the optional data parameter:<br><br>`btn.on("click", function(evt, customerNumber) {alert(customerNumber); }, 1234);`<br><br>In this case, the additional data, `1234`, is passed to the custom function through the customerNumber parameter.<br><br>For a further example of using the on() function, see *Defining an Event Handler for a Button*. |
| setImage(url) | this | Yes | 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. | `btn.setImage("http:// domain/ imgs/img.gif");`<br><br>or<br><br>`btn.setImage("/images/ test.png");` |
| setParent(tb) | this | Yes | Sets the given TitleBar as the parent of this button.<br><br>You can create a button first, without giving it a parent. You can then call this method to add the button to a TitleBar.<br><br>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. | `btn.setParent(titlebar);` |
| setText(text) | this | Yes | Sets the text of this button. No text is displayed if a null or empty string is passed. | `btn.setText("Text is Changed");` |
| setToolTipText(tool TipText) | this | Yes | Sets the tooltip text of the button. If a null or empty string is passed, no tooltip is set for this button. | `btn.setToolTipText ("Tool Tip Text for Button");` |

| Method Name | Return Type | Chain-able | Description | Sample Code |
|---|---|---|---|---|
| | | | **Note:** If you use oraclecrmod.enableIdHint() in your code, the tooltip for enableIdHint() is displayed on top of the tooltip text for setToolTipText(). | |
| setToolTipFormat() | this | Yes | Sets the tooltip format of the button, which can be either HTML or plain text. If a null or empty string is passed, the format is set as `Plain Text`. The default value is `Plain Text`. | `btn.setToolTipFormat ("HTML");`<br><br>For more information about tooltip formatting, see *Considerations for Specifying Tooltips with HTML Formatting*<br><br>. |
| show() | this | Yes | Shows the button if the button is currently hidden. | `btn.show();` |

# 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:

```
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:

```
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:

## To find the ID of a title bar or button

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 the Field Object

The following table describes the methods that are available for the Field object. These methods are used in the getting and setting of field values and in the color coding of field colors.

For information about finding the field names that you must use when working with these methods, see *Finding Field Names*.

| Method Name | Return Type | Description | Notes | Sample Code |
|---|---|---|---|---|
| getField(fieldName) | A field object exposing the getValue and setValue methods or null, if the field is not found on the screen | Gets the field on the current screen that is identified by the fieldName parameter. | For information about the ListRow getField() method, see the table in *Methods for the ListRow Object*. | `oraclecrmod.getField('Parent Account Name');` |
| getValue() | The field value string | Gets a value for a field in the current screen. | For a check box field, the return values are `Y` for selected, and `N` for deselected. | `oraclecrmod.getField('Location').getValue();` |
| getLICValue() | The field's language independent code (LIC) value as a string | Gets the LIC value for a field in the current screen. | For more information, see *Getting and Setting LIC Values of Picklists*. | `oraclecrmod.getField("Type").getLICValue();` |
| setValue (fieldValue) | The new field value string that was just set, or null if the method fails | Sets a value for a field in the current screen. This can be a New page, an Edit page, a Detail page, or a List page. | For information about the limitations of setValue(), such as field types not supported, see *Limitations When Using the setValue() Method*. | For a New, Edit, or Detail page:<br>`oraclecrmod.getField('Type').setValue("Customer");` |

| Method Name | Return Type | Description | Notes | Sample Code |
|---|---|---|---|---|
| | | | **Note:** On Detail pages, and for lists, setValue() does not commit values to the database and update the screen. The commitValues() method must be called to commit the values. See the tables in *Methods for the Form Object* and *Methods for the List Object*. | For a list, including a related item list:<br>`oraclecrmod.getList ("AccountList").getRow(0).getField ("Type").setValue(" Customer");` |
| setLICValue(fieldValue) | The new language independent code (LIC) field value string that was just set, or null if the method fails | Sets a LIC value for a field in the current screen. | The LIC value is used to set the field's value. The language dependent code (LDC) value will still be displayed on the screen.<br>For more information, see *Getting and Setting LIC Values of Picklists*. | `oraclecrmod.getField('Type'). setLICValue("Customer");`<br>For a Detail page:<br>`oraclecrmod.getField("Priority"). setLICValue("Low"); oraclecrmod.getField("Region"). setLICValue("West"); oraclecrmod.getForm(). commitValues(commitValuesHandler );` |
| setColor(JSONObject) | this | Sets the colors for a Field by passing in a JSONObject. | Elements supported in the JSONObject:<br>• **labelBgColor**. The field label background color.<br>• **labelTextColor** The field label text color.<br>• **valueBgColor**. The field value background color.<br>• **valueTextColor**. The field value text color (not applicable on Edit pages).<br>Any other elements are ignored.<br>For information about the color values the elements can have, see *Color Values for JSONObjects*. | `field.setColor({" labelBgColor":"Red", "labelTextColor": "Yellow", "valueBgColor":"Yellow", "valueTextColor":"Red"});` |
| getColor() | JSONObject | Returns the JSONObject that was set for color coding the field. | If no JSONObject was set using setColor(), getColor() returns the colors based on the system or theme colors.<br>The getColor() method always returns RGB values or "transparent".<br>On Edit pages, text color values are not returned. | `color = field.getColor();` |

| Method Name | Return Type | Description | Notes | Sample Code |
|---|---|---|---|---|
| | | | Text color values are not returned for field types that are not text based, such as check boxes or visual indicators. | |
| getId() | String | Returns the row ID of an associated record based on the associated record field displayed on the screen. | The getId() method is supported on Detail, Edit, and List pages. For New pages, it returns null.<br><br>The getId() method on a field only works for associated record fields and returns null for other types of fields.<br><br>Multi-association fields are not supported, for example, the Users and Contacts fields of Activity. | For example, if an Account's Primary Contact field is on the screen, then to return the Primary Contact ID value:<br><br>`getField("Primary Contact").getId();`<br><br>For an associated record on a List page:<br><br>`oraclecrmod.getList().getRow(0).getField("Owner").getId();` |

**Note:** For fields like Modified that have a user name link and a date, getColor() returns the user name link color. However, setColor() can set the color on both the link and text.

# 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 Type 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.

## To find field names using browser developer tools

1. Right click in the field for which the name is required.
2. Select the appropriate option, 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. For New or Edit pages, 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

4. For Detail and List pages, check the id attribute value of the <td> element containing the field, which is in the format AX.RY.FieldName, for example, `A0.R1.Location`. In the attribute value, X is the applet index and Y is the row index. For Detail forms, the prefix is always `A0.R0`, while for full lists, the prefix is `A0`. RY. The part after the last dot, for example, `Location`, is the field name.

**Note:** For check boxes and multi-select 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.

# Getting and Setting Screen Values

You can use the getValue() and setValue() methods to get and set values for fields on the current screen. The methods are supported for New and Edit pages, and also for Detail pages (including related information sections) and lists (including related item lists). For information about the getValue() and setValue() methods, see the table in *Methods for the Field Object*.

Using the setValue() method on Detail pages or List pages is similar to inline editing in the Oracle CRM On Demand UI. When you use the setValue() method for Detail pages and List pages, the value is not set and updated in the database immediately. You must use a commitValues() method to commit the setting of values on Detail or List pages. For information about the commitValues() method, see the tables in *Methods for the Form Object* and *Methods for the List Object*.

For information about finding the field names that you must use when working with these methods, see *Finding Field Names*.

## Limitations When Using the setValue() Method

The setValue() method is not supported for the following types of field:

- Analytics fields
- Color fields (that is, fields used for color definitions in themes)
- Currency code fields
- Multi-association fields
- Shared address fields
- Single-association fields for which the Auto-Resolve Enabled check box is deselected. Such fields are grayed out on the UI.

If you call setValue() for the above field types, null is returned.

Single-association fields and fields such as Sales Stage (on Opportunity) only support setting by row ID on Detail pages.

Concatenated fields are not supported, but you can set the values for constituent fields of concatenated fields for both Detail and Edit pages, regardless of whether the fields are on the page layout.

For a check box field, only the values `Y` or `N` are accepted for the input parameter. Any other values are ignored.

### Support for Address Fields When Using the setValue() Method

For addresses on both Edit and Detail pages, you can change the Country field using the setValue() method and at the same time set other fields in the address layout relevant to the new country setting.

The field names for address fields have the format:

```
ParentAddressName.ChildAddressName
```

For example, for an Account billing address Country field:

```
Parent Bill To Address.Bill To Country
```

As a code example, you might change the country from Canada to USA, then set the state as follows:

```
oraclecrmod.getField("Parent Bill To Address.Bill To Country").setValue("USA");
oraclecrmod.getField("Parent Bill To Address.Bill To State").setValue("NY");
oraclecrmod.getForm().commitValues(Handler);
```

## 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.

## Getting and Setting LIC Values of Picklists

You can use the getLICValue() and setLICValue() methods to respectively get and set language independent code (LIC) values for picklists as opposed to the language dependent code (LDC) values that are displayed on the screen.

You can use the getLICValue() and setLICValue() methods for New, Edit, Detail, and List pages (including related item lists). On Detail pages, and for lists, setLICValue() does not commit values to the database and update the screen. The commitValues() method must be called to commit the values. For information about the commitValues() method, see the following and the table in *Methods for the List Object*.

You can use the getLICValue() and setLICValue() methods for preconfigured picklists, custom picklists, and cascading picklists are supported, but multi-select picklists are not supported. Calling the getLICValue() and setLICValue() methods on multi-select picklists or on non-picklist fields results in the same behavior as for the getValue() and setValue() methods, that is, the displayed (LDC) value is returned.

As an example, the Priority picklist on the Opportunity record type has LIC values: Low, Medium, High, which are displayed in French as Faible, Moyen, Eleve.

Therefore, if the user's language is French, and the Priority picklist field is set to Low:

- getValue() returns `Faible`
- getLICValue() returns `Low`

For some preconfigured picklist fields such as Industry (on the Account record type), Sales Stage (on Opportunity), and Role (on various record types) the LIC values are row ID values.

For these fields the getLICValue() method retrieves the row ID of the selected value for the picklist. For example:

- Industry.getValue() returns `High Technology`
- IndustryField.getLICValue() returns `1QA2-11RH89`

The setLICValue() method allows you to set the picklist value using a row ID, for example:

```
field.setLICValue("ROWID")
```

When set, the on-screen value is updated to the display value associated with the row ID.

You can determine the row ID for each value in the fields by using browser developer tools as described in *Finding Field Names*.

## Color Values for JSONObjects

For the setColor() methods used in color coding of fields, the following types of color value can be specified for each element in the JSONObject:

- The name of a color, for example: `Red, Yellow, Black`. The values are not case sensitive.
- A hexadecimal number for a color, for example: `#FF0000`, the number for red.
- An RGB value for a color, for example: `rgb(0, 0, 255)`, the value for blue.

# Methods for the Form Object

The following table shows the methods that are available for the Form object. These methods are used, together with the field.setColor() and field.getColor() methods in the color coding of fields in Detail and Edit pages. For more information about the setColor() and getColor() methods, see the table in *Methods for the Field Object*.

| Method Name | Return Type | Description | Sample Code | Notes |
|---|---|---|---|---|
| on(event,customfunction) | this | Registers a custom handler for the form. | `form.on("display", myColorHandler);` | The custom handler is automatically executed when the form initially loads or refreshes due to any other actions on the screen such as inline editing.<br><br>The event parameter must be "display". If not, it is ignored and the custom handler is not registered. |
| off(event) | this | Removes the custom handler for the form. | `form.off("display");` | If the event parameter is not "display", it is ignored and the custom handler is not removed. |
| commitValues(callback) | JSONObject in callback | Commits values set on Detail pages using the setValue() method. | `oraclecrmod.getForm().commitValues(callback);` | For information about the callback handler, see *Callback Handler for the commitValues() Method*. |

| Method Name | Return Type | Description | Sample Code | Notes |
|---|---|---|---|---|
| getId() | String | Returns the row ID of the record on the screen. | `oraclecrmod.getForm().getId();` | The getId() method is supported for Detail and Edit pages. For other types of pages, including New pages, it returns null. |

# Example of a Custom Handler for a Form Object

The following is a pseudocode example of a custom handler for a Form object. The example includes an RGB value (rgb(255,255,0)) for the field value background color of yellow, and a hexadecimal value (#ff0000) for the field value text color of red.

```
function myColorHandler()
{
 var accountValue = oraclecrmod.getField("AccountValue").getValue();
 var field = oraclecrmod.getField("AccountName");

 if (accountvalue>1000000) then
 {

 field.setColor({"labelBgColor":"Red","labelTextColor":"Yellow",
 "valueBgColor":"rgb(255,255,0)","valueTextColor":"#ff0000"});

 }
 else {
 field.setColor({"labelBgColor":"","labelTextColor":"",
 "valueBgColor":"","valueTextColor":""});

 }
}
//set up the color handler
oraclecrmod.getForm().on("display",myColorHandler)
```

For a full code sample, see *Code Sample for Color Coding of Fields and Rows*.

# Callback Handler for the commitValues() Method

When the commitValues() method completes processing, and if there is an error, a custom error callback handler is called.

The callback handler returns a JSONObject, which has the following properties:

- **status**. The status as a string: `"OK"` or `"ERROR"`.
- **errors**. Any errors in a JSONArray object. The errors object specifies null or an array of objects in the following format:

```
{
 fac : "error code",
 msg : "error message"
}
```

There are two types of error:

1. Errors where validation failed on the JavaScript side.

In this case, the error code is one of:

- OCCAM_JS_INLINE_FIELD_REQUIRED
- OCCAM_JS_INLINE_TEXT_TOO_LONG
- OCCAM_JS_INLINE_INVALID_FORMAT

The error message has the format: %Field_Name% : %Failed_Reason%

2. Errors returned from the OM side.

   In this case, the error code has the format SLB-XXX-XXXXX, and the error codes and messages are similar to those described in *CRUD Error Codes and Messages*.

For information about handling errors in your code, see *Errors and Error Handling*.

The callback handler is not only an error handler, it is also where your logic resumes.

# Methods for the List Object

The following describes the methods that are available for the List object. These methods are used for color coding in lists.

| Method Name | Return Type | Description | Sample Code | Notes |
|---|---|---|---|---|
| getRow(Row#) | ListRow | Returns the specific row in the displayed list based on the Row# parameter. | `listRow= listObject.getRow(0);` | The first row has index 0 and the last row has index getDisplayedCount()-1.<br><br>If the Row# value is invalid, then null is returned. |
| getDisplayedCount() | Integer | Returns the total number of rows displayed on the screen. | `x = listObject.getDisplayedCount();` | None |
| on(event, customhandler) | this | Registers a custom handler for the list currently displayed on the screen.<br><br>**Note:** The framework passes the custom handler a row parameter, see *About the Custom Handler for a List Object*. | `functionmyColorHandler(row){`<br>`//Change column YYY's color based on column XXX's value`<br>`if (row.getField("XXX").getValue() =="High"){`<br>`row.getField("YYY").setColor(...);`<br>`}`<br>`//Special logic for first row`<br>`if (row.rowNum == 0){`<br>`row.setColor(...);`<br>`}`<br>`}`<br>`listObj.on("display", myColorHandler);` | For initial page load and paging, the handler is called for each row on the list.<br><br>For inline editing, the row object that is passed to the custom handler is based on the row on which the inline edit was processed.<br><br>The event parameter must be "display". |

| Method Name | Return Type | Description | Sample Code | Notes |
|---|---|---|---|---|
| off(event) | this | Removes the custom handler | `listObj.off("display");` | If the event parameter is not "display", it is ignored and the custom handler is not removed. |

# About the Custom Handler for a List Object

The custom handler for a List object has the format colorFunction(row). The JavaScript API framework provides the row parameter, corresponding to a ListRow object, when calling the custom function.

For inline editing, the row object corresponds to the row in which the inline editing occurs.

For initial page load and paging, the custom handler is called for each row in the list with the same logic applying for each row. For example, for a list of 25 rows, the JavaScript API framework calls the custom handler 25 times, providing the row parameter with values 0 through 24 (that is row# 0 through row count -1). Rather than looping, you only need to focus on the logic for row criteria and color setting. If you need to apply some special logic for a particular row, you can check the listRow.rowNum property.

The following is an example of pseudocode for a custom handler for a List object:

```
function otherColorHandler(row) {
if (row.getField("Location").getValue() = "Hometown") then
{
row.getField("Location").setColor("valueBgColor":"Black","valueTextColor":"Yellow");
}
}


listA = oraclecrmod.getList("AccountList");

listA.on("display",otherColorHandler);
```

For a full code sample, see *Code Sample for Color Coding of Fields and Rows*.

# Finding List Names

You can use recent versions of browsers to find the name of lists on List and Detail pages.

## To find list names using browser developer tools

1. Right click on the list for which the name is required.
2. Select the appropriate option, for example, Inspect element or Inspect with.

   The name of the option varies with browser versions. The browser opens developer tools and displays the HTML source.
3. Check the id attribute value of the <table> element containing the list, which is in the format listnameLIST, for example, AddressChildListLIST. The part before LIST, for example, AddressChildList, is the list name.

# Methods for the ListRow Object

The following table describes the methods that are available for the ListRow object. These methods are used for color coding in lists:

| Method Name | Return Type | Description | Notes | Sample Code |
|---|---|---|---|---|
| setColor(JSON Object) | this | Sets the colors for the row by passing in a JSONObject. | Elements supported in the JSONObject:<br>• rowBgColor. The row's background color.<br>Any other elements are ignored.<br>For information about the values the element can have, see *Color Values for JSONObjects*. | `listRow.setColor({"rowBgColor":"Black"});` |
| getColor() | JSONObjec | Returns the row's RGB value in a JSONObject. | The getColor() method always returns the row's background color in RGB values or "transparent". | `color = listRow.getColor();` |
| getField(fieldname) | Field | Returns a specific field object in the row based on the field name. | The only methods of the Field object available when used in a list are:<br>• getValue()<br>• setColor() and getColor() but labelBgColor and labelTextColor do not apply.<br>For more information about these methods, see the table in *Methods for the Field Object*. | `listRow.getField("Account Name");` |
| rowNum | N/A | Property containing the row number. | None | `if (row.rowNum == 0){ row.setColor(...);` |
| getId() | String | Returns the row ID of the record in a row in a list on the screen.<br>Related item lists are also supported. | In some related item lists, for example, for Account Contact, `getRow(#).getId()` returns the intersection row ID. In this case, to get the ID of the actual contact associated to the account through the Contact child list, you must call `getRow(#).getField` | `oraclecrmod.getList().getRow(0).getID();` |

| Method Name | Return Type | Description | Notes | Sample Code |
|---|---|---|---|---|
| | | | `("Contact Full Name").getId();` | |
| commitValues(callback) | JSONObject in callback | Commits field values set on list pages using the setValue() method | For information about the callback handler, see *Callback Handler for the commitValues() Method*. | `oraclecrmod.getList("AccountList") .getRow(0).commitValues(callback );` |

# 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 the following table are helper functions that wrap the attributes for the context object shown in the second table. 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()
```

| Method Name | Description | Parameters and Notes |
|---|---|---|
| isHomePage | Returns whether the current page type is Homepage | None |
| isListPage | Returns whether the current page is a List page | None |
| isDetailPage | Returns whether the current page type is Detail page | None |
| isEditPage | Returns whether the current page type is Edit page | None |
| isNewPage | Returns whether the current page type is New page | None |
| isEditOrNewPage | Returns whether the current page type is Edit page or New page | None |
| isAdminPage | Returns whether the current page type is Admin page | None |

| Method Name | Description | Parameters and Notes |
|---|---|---|
| | | |
| isObject(object) | Returns whether the current page's primary object is the record type in the parameter | The String object parameter is the record type, for example: `oraclecrmod.ctx.isObject("Account")` |

All of the helper methods have a Boolean return type. The following table shows the attributes for the context object.

| Attribute | Type | Description | Sample Values and Code |
|---|---|---|---|
| servlet | String | The current servlet path. | `oraclecrmod.ctx.servlet == "/user/AccountHomePage"` |
| pageType | String | An alphabetic character representing the current page type. | `H` for Homepage, `L` for List page, `D` for Detail page, `N` for New page, `E` for Edit page, and `A` for Admin page.<br><br>`if (oraclecrmod.ctx.pageType=="D") {...}` |
| object | String | The current record type name. | `"Account", "CustomObject4"` |
| layoutId | String | 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. | `"1QA2-P4F9O", "Lead"` |
| layoutName | String | The display name for the current layout. | `"myAcctLayout"` |
| roleId | String | The row ID for the current user's role, which you can find in the URL of the Role Management Wizard editing page. | `"1-G4WZO"` |
| roleName | String | The name of the current user role, unlocalized | `"Administrator"` |
| lang | String | The current user's language as a three-character code<br><br>**Note:** See *Oracle Migration Tool On Demand Guide* for a list of the language codes supported for Oracle CRM On Demand. | `"enu", "deu"` |

| Attribute | Type | Description | Sample Values and Code |
|---|---|---|---|
| userId | String | The user ID of the currently logged-in user. | `"1QA3-HTE12"` |
| userCountryLIC | String | The country (LIC value) of the currently logged-in user. | `"Korea"` |
| userEmail | String | The email of the currently logged-in user. | `"mini.me@acme.com"` |
| userAlias | String | The user alias of the currently logged-in user. | `"minime"` |
| userFullName | String | The full name of the currently logged-in user. | `"Mini Me"` |

# Methods for CRUD Operations

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

For the CRUD operations, only top-level record types are supported and not child-level record types. To access child-level record types, you can use the Oracle CRM On Demand REST API. For more information, see *Oracle CRM On Demand REST API Developer's Guide* .

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

> **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.

| Method Name | Return Type | Description | Sample Code |
|---|---|---|---|
| createRecord (objectName, fieldNameValuePairs, createParameters, callback) | JSONObject | 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. | `oraclecrmod.dataSvc.createRecord ("Account", {"Name" : "Account 1", "Account Contact Role" : "Admin", "Type": "Customer", "Description" : "Description Test"}, null, callback);` |
| readRecord (objectName, fieldNames, readParameters, callback) | JSONObject | 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.<br><br>**Note:** Only row ID values are supported for the searchType parameter. | `oraclecrmod.dataSvc.readRecord ("Account", "Name, Description", { searchType: "rowId", "rowId": "1QA2-TNRWW"}, callback);` |
| updateRecord (objectName, fieldNameValuePairs, updateParameters, callback) | JSONObject | 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.<br><br>**Note:** Only row ID values are supported for the searchType parameter. | `oraclecrmod.dataSvc.updateRecord ("Account", {"Name" : "Account 1", "Account Contact Role" : "Admin", "Type": "Customer", "Description" : "Description Test"}, {searchType: "rowId", "rowId": "1QA2-TNRWW"}, callback);` |
| deleteRecord (objectName, deleteParameters, callback) | JSONObject | 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.<br><br>**Note:** Only row ID values are supported for the searchType parameter. | `oraclecrmod.dataSvc.deleteRecord ("Account", "1QA2-TNRWW", callback);` |

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 the table in *Methods for CRUD Operations*:

- **objectName**. The name of the record type, for example, `"Account"`
- **fieldNames (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"]
  ```

- **fieldNameValuePairs (for createRecord and updateRecord)**. An object containing field name-value pairs. For example:

  ```
  {"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:

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

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

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

  > **Note:** Only row ID values are supported for the searchType parameter. To retrieve all searchable fields, you can use the REST API instead. Note that the REST API uses integration tag names rather than the HTML field tag names used by the JavaScript API. For more information, see *Oracle CRM On Demand REST API Developer's Guide* .

- **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*.

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:

```
[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:

```
[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*.
- 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 the following.

| Method Name | Description | Sample Code |
|---|---|---|
| Callback function name | callback (request, response)<br><br>User-defined JavaScript function to process the results of the CRUD operations. | `callback(request,response)`<br>`{`<br>`}` |

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:

```
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*.

# Helper Functions for Callback Functions

The following describes the helper functions that are available for callback functions.

| Method Name | Return Type | CRUD Operations | Description | Sample Code |
|---|---|---|---|---|
| getRowId() | row ID | Create, read, update | Gets the row ID of the record in the response object. | `var rowID = response.getRowId();` |
| getModId() | mod ID | Create, Read, Update | Gets the mod ID of the record in the response object. | `var modId = response.getModId();` |
| getFieldValue (fieldName) | Field value | Read | Gets the field value of the corresponding field name passed in the function. | `var desc = response.getFieldValue ("Description");` |

# Sample Code for Callback Function

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

```
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* for an example of a callback function that performs error handling.

# Method for Setting the Search Specification for the Solutions Popup Window Associated with Service Requests

You can use the setPopupSearchSpec() method to specify the search specification for a popup window as shown in the following.

**Note:** The setPopupSearchSpec() method is only supported for the multiassociation Lookup window for solutions associated with a service request.

| Method Name | Description | Limitations |
|---|---|---|
| setPopupSearchSpec (popUpBtn, searchSpec) | Adds a search specification parameter in the HTTP POST request for a popup window. The method adds the parameter if it does not exist, and updates it if the parameter already exists. | For searchSpec, only the AND operation and String and Picklist fields are supported. |

The parameters for the setPopupSearchSpec() method are as follows:

- **popUpBtn**. The Add button that triggers the popup. You can use the getButton() method to obtain the button, for example:

  ```
  var addBtn = oraclecrmod.getButton('BTN_TB_SolutionChildList_AddButton')
  ```

- **searchSpec**. The search specification for filtering the results returned in the Solutions multiassociation Lookup window. The following shows an example of a search specification:

  ```
  var searchSpec = "Name='Training' and Status='Approved'"
  ```

  The length of the search specification must be less than 4096 characters.

The following is an example of how you can use the setPopupSearchSpec() method. The sample code creates a filter for the multiassociation Lookup window that is displayed on clicking the Add button in the Solutions section on the Service Request Detail page. The search specification filters for solutions where the values of the Area, Sub Area 1, and Sub Area 2 fields are equal respectively to the values of the Area, Sub Area 1, and Sub Area 2 fields of the Service Request.

```
oraclecrmod.onReady(function()
{

 if (oraclecrmod.ctx.isObject("Service Request") && oraclecrmod.ctx.isDetailPage())

 {

 // Define function that sets searchSpec of Solutions popup window
 function addMultiAssocSearchSpec()

 {
 // Get the Area, Sub Area 1, Sub Area 2 fields of the Service Request
 var SRArea = oraclecrmod.getField('Area');
```

```
 var SRSubArea1 = oraclecrmod.getField('ZPick_0');
 var SRSubArea2 = oraclecrmod.getField('ZPick_1');


 if (SRArea != null && SRSubArea1 != null && SRSubArea2 != null)
 {

 // ZPick_0, ZPick_1, Z_Pick_2 are the HTML tags of fields Area, Sub Area
 // and Sub Area 2 for Solution respectively
 setPopupSearchSpec(this, "ZPick_0='" + SRArea.getValue() + "' AND ZPick_1='" + SRSubArea1.getValue() + "'
 AND ZPick_2='"+ SRSubArea2.getValue()
 +"'");
 }
 }
 // Find the Add Solution popup button in the Service Request Page
 var addBtn = oraclecrmod.getButton('BTN_TB_SolutionChildList_AddButton');
 if (addBtn != null)
 {

 // When the mouse is over the button update the searchSpec for the button
 addBtn.on ("mouseover", addMultiAssocSearchSpec);

 }

 }

});
```

# Errors and Error Handling

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

```
{

 fac : "SBL-ODU-MSG_NUMBER", // error code
 msg : "MSG_TEXT" // error message

}
```

For example, for response.errors[0]:

```
{
 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 handing 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.

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

A guideline is to use the error code in error handling rather than the error message. This is recommended because the error code is language independent, but the error message is not. Code like the following only works in English:

```
if (response.errors[0].msg == "No records found")
```

# CRUD Error Codes and Messages

The following shows some error codes and messages that might be returned.

| Code | Message | Description |
|---|---|---|
| SBL-ODU-00251 | The requested operation has encountered an error. | An internal configuration error occurred. This error is rare. |
| SBL-ODU-00271 | Invalid value for the parameter: parameter_ name. | The parameter for the CRUD operation specified a null or empty objectName parameter, a nonexistent objectName (for example, "Acount"), or a null or empty row ID. |
| SBL-ODU-57081 | No records found. | An operation tried to read a nonexistent record. |
| SBL-DAT-00398 | Field 'field_name' does not exist in definition for business component 'record_type'. Please ask your systems administrator to check your configuration. | An operation specified a nonexistent field name. |
| SBL-DAT-00498 | '<field>field_name</field>' is a required field. Please enter a value for the field. | An operation did not specify a value for a field that is required. |
| SBL-ODS-50006 | 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. | An operation tried to create a record that already exists. |

# 5 JavaScript API Code Samples

## JavaScript API Code Samples

This appendix contains code samples for the public JavaScript APIs available with Oracle CRM On Demand. It includes the following topics:



## Getting Started with the Code Samples

To use the sample code in this appendix:

1. Log in to Oracle CRM On Demand with an administrator role that includes the privilege: Upload Client Side Extensions and Manage Custom HTML Head Tag.
2. Navigate to Admin, Application Customization, Custom HTML Head Tag.
3. In the Custom HTML Head Tags Detail page, add the sample code.

For more information about getting started, see *Getting Started with the JavaScript API* and in particular see *Privileges Required* and *Managing HTML Head Additions*.

For each of the code samples, the entry point for running the custom code is either a call to the onReady() method, which is associated with the ready event, or the onLoad() method, which is associated with the onload event. The ready event occurs after the HTML document has been loaded, while the onload event occurs later, when all content (for example, images) has also been loaded.

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

# Code Sample 1 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.

```
// 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 nextstep = oraclecrmod.getField("Next Step").getValue();
 // validate custom business logic goes here based on field values retrieved

 }


 // 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 2 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 field. The sample code also hides the Add button on the Contact related information applet.

```
// entry point for running custom code
oraclecrmod.onReady(function()
{
```

```
// OPPORTUNITY VALIDATE EXAMPLE //
// 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();
// convert to currency string to number
var number = Number(revenue.replace(/[^0-9\.]+/g, ""));
// validate custom business logic goes here based on field values retrieved
if (number > 0) {

var alertStr = "Recommendation is to fill in the field(s):\n";
alertStr += "Next Step\n\n";
alertStr += "Based on the rule(s):\n";
alertStr += "Revenue is greater than 0\n\n";
alertStr += "For this record:\n";
alertStr += "Revenue is: " + revenue;
alert(alertStr);

}
}

// 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 += 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 Creates a Child Record

This following code sample creates a custom button labeled `Create` Task on the Opportunity Detail page. When the button is clicked, the REST API is used to create a child Activity record. In the createTask() function, you must replace <PodURL> with the URL for your pod. The Restful Services Integration privilege is required to send REST requests.

```
function createHttpRequest (httpmethod)

{

var xmlhttpRequest = null;
xmlhttpRequest = new XMLHttpRequest();
if (typeof xmlhttpRequest.overrideMimeType != 'undefined')

{

xmlhttpRequest.overrideMimeType('application/json');
}

return xmlhttpRequest;
```

```
}


 oraclecrmod.onReady(function()
 {
 // when on the Opportunity Detail page
 if(oraclecrmod.ctx.object == "Opportunity" && oraclecrmod.ctx.isDetailPage())
 {

 function createTask()
 {

 var id = oraclecrmod.getField("Id").getValue();
 var insertPayload = "{\"Activities\":[ \
 {\
 \"Activity\": \"Appointment\", \
 \"Subject\": \"Opp Auto Appointment\", \
 \"Type\": \"Other\", \
 \"Location\": \"Markham\", \
 \"OpportunityId\":\"" + id + "\" \
 }\
 ]}"


 //Calls REST API to create Task
 var url = "<PodURL>/OnDemand/user/Rest/026/Activities";
 var req = createHttpRequest ("POST");
 req.open('POST', url, false);
 req.setRequestHeader("Content-Type","application/
 vnd.oracle.adf.resource+json");
 req.send(insertPayload);
 alert(req.responseText);
 //success
 if (req.readyState == 4 && req.status == 201)
 {

 alert("success");

 }
 }
 // get the title bar
 titleBar = oraclecrmod.getTitleBar("OpportunityFormTB");
 // create the new button
 button = oraclecrmod.createButton({id:"CreateTaskButton", text:"Create Task",parent:titleBar});
 // associate the createTask function with the button click event button.on("click", createTask);

 }

 }

);
```

# 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(
```

```
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.

```
// 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 To Street Address 2").getValue() + " ";
 if (oraclecrmod.getField("Ship Street Address 3") != null)
 wholeAddress += oraclecrmod.getField("Ship To Street Address 3").getValue() + " ";
 if (oraclecrmod.getField("Ship To County") != null)
 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);


 }
});
```

# Code Sample for a Custom Button That Creates a Task

The following sample code creates a custom button labeled Assign Next Step Task on the Opportunity Detail page. When the button is clicked, a function is called to open the Task Open page with appropriate values. The task can then be saved and used to follow up on the next step for the opportunity.

```
// entry point for running custom code
oraclecrmod.onReady(function()
{

 // when on the Opportunity Detail page
 if (oraclecrmod.ctx.object == "Opportunity" && oraclecrmod.ctx.isDetailPage()) {
 // define createTask function
 function createTask() {
 var Id = oraclecrmod.getField("Id").getValue();
 var oppName = oraclecrmod.getField("Name").getValue();
 var nextstep = oraclecrmod.getField("Next Step").getValue();
 var desc = "Please follow up on the next step for this opportunity: " + nextstep;
 var subject = "Follow up on next step for " + oppName;
 var date1 = new Date();
 date1.setDate(date1.getDate() + 2);
 var dd = date1.getDate();
 var mm = date1.getMonth()+1; // month starts at 0 e.g. Jan = 0
 var y = date1.getFullYear();
 var date1Formatted = mm + '/' + dd + '/' + y;

 window.open("/OnDemand/user/
 TaskNew?OMRET0=OpportunityDetail%3fOpptyDetailForm.Id%3d" + Id + "&OMCR0=" + Id +
 "&OCTYPE=&OMTGT=TaskEditForm&OMTHD=ActivityNewNav&OMCBO=Opportunity&TaskEdit
 Form.Due Date=" + encodeURIComponent(date1Formatted) +
 "&TaskEditForm.Comment=" + encodeURIComponent(desc) +
 "&TaskEditForm.Description=" + encodeURIComponent(subject), "_self");

 } //createTask


 // get the title bar
 titleBar = oraclecrmod.getTitleBar("OpportunityFormTB");



 // create the new TASK button
 button = oraclecrmod.createButton({
 id: "createTaskButton",
 text: "Assign Next Step Task",
 parent: titleBar
 });


 // associate the createTask function with the button click event
 button.on("click", createTask);

 }
});
```

# Code Sample for Hiding a Button

The following sample code hides the copy button on the Opportunity Detail page.

```
oraclecrmod.onReady(function()
{
 // when on the Contact Detail page
 if(oraclecrmod.ctx.object == "Opportunity" && oraclecrmod.ctx.isDetailPage())
 {
 // get the Copy button
 button = oraclecrmod.getButton("BTN_TB_OpportunityForm_OpportunityPreCopyNav");


 // hide the button
 button.hide();

 }
});
```

# Code Sample for Changing the Behavior of a Save Button

The following code customizes the Save button on the Account Edit page so that the code changes the owner of the record:

```
oraclecrmod.onReady(

function(){

if(oraclecrmod.ctx.isObject("Account") && oraclecrmod.ctx.isEditPage()){

 var btnSave = oraclecrmod.getButton("BTN_TB_AccountEditForm_Save");

 if(btnSave != null){

 btnSave.offAll("click").on("click",function(){

 //set on screen value api to set the value.

 oraclecrmod.getField('Owner Alias').setValue("User1");

 btnSave.invokeDefault("click");
 });

 }

 }

});
```

# Code Sample for a Read Operation on an Account Record

The following sample code is for a read operation to get details of an account and display them in the Description field on the Contact Edit page. A call is made to the readRecord() method to get the values for the Type and Location fields of the associated account.

```
// entry point for running custom code - only before the page loads
oraclecrmod.onLoad(function () {
 // CONTACT EXAMPLE //
 var contactCallback = function (request, response) {
 if (response.status == "OK") {
 var type = response.getFieldValue("Type");
 var location = response.getFieldValue("Location");
 //update the description field with account record info
 oraclecrmod.getField('Comment').setValue("This contact is created by the
 account with type " + type + " and located at " + location);
 }
 }

 // when on the Contact new page
 if (oraclecrmod.ctx.object == "Contact" && oraclecrmod.ctx.isNewPage()) {
 //get the Account Id
 var acctId = oraclecrmod.getField("Account Id").getValue();
 if (acctId != null) {
 oraclecrmod.dataSvc.readRecord("Account", "Location,Type", {
 searchType: "rowId",
 "rowId": acctId
 }, contactCallback);
 }
 }
});
```

# Code Sample for Color Coding of Fields and Rows

The following code sets colors on pages as follows:

- On the Opportunity Detail or Edit page, the colors of the label and value for the Primary Revenue Win Probability field are set, depending on the percentage value entered in the field.
- On the Opportunity List page, the colors for the Primary Revenue Win Probability field are set, depending on the percentage value entered in the field, and rows where the Priority field has the value High are colored yellow.
- On the Contact related item list of the Opportunity page, the color of the Role field is set, if the value of the field is `Decision Maker`.
- On any page where there is a Name field, the field label background color is set to black, the field label text color to red, the field value background color to red, and the field value text color to black.

```
//color function for coloring the Opp detail/edit page
function colorOppty()

{
```

```
 //on the Opportunity detail/edit page, change the color for the Probability field depending on the
 percentage entered

 var percent= oraclecrmod.getField("Primary Revenue Win Probability");
 colorPercent(percent);

}
//color function for coloring the Opp List
function colorOpptyList(row)
{

 var percent= row.getField("Primary Revenue Win Probability");
 var priority = row.getField("Priority");
 colorPercent(percent);
 if (priority != null)
 {

 //set the row to Yellow in the list if the priority is High
 if (priority.getValue()=="High")
 {

 row.setColor({rowBgColor:"yellow"});
 } else {
 row.setColor({rowBgColor:""}); //this clears previous set color and will display based on theme color

 }
 }
}
//common color function to color the Probability field in the same way in the list page as it is colored on
 the detail/edit page
function colorPercent(percent)
{
 var myred = "#e71832";
 var myblue = "rgb(66,97,156)";
 var myyellow= "rgb(255,215,181)";
 if (percent != null)
 {

 if(percent.getValue()> 50)

 {

 percent.setColor({"labelBgColor":myred,"labelTextColor":myblue,"valueBgColor":myblue,"valueTextColor":myred});

 }
 else {

 percent.setColor({"labelBgColor":"","labelTextColor":"","valueBgColor":"","valueTextColor":""}); //this
 clears previous set color and will display based on theme color
 }
 }
}
//color function for coloring the Opp's Contact Child List
function colorOpptyContactChildList(row)

{

 var role = row.getField("Role");
 if (role != null)
 {
 //set the row to Cyan in the list if the contact's role is Decision maker
 //change text of role to Red as well
 if (role.getValue()=="Decision Maker")
 {
```

```
 row.setColor({rowBgColor:"cyan"});
 role.setColor({valueTextColor:"Red"});
 }
 else {
 row.setColor({rowBgColor:""}); //this clears previous set color and will display based on theme color

 role.setColor({valueTextColor:""}); //this clears previous set color and will display based on theme color

 }

 }

}
// entry point for running custom code
oraclecrmod.onReady(function()
{
 //if Name field exists on the form, then set the field label background to Black, field label text to Red,
 value background to red and value text to black

 //for example, this applies to Account Name on Account detail/edit page and Opportunity Name on Opportunity
 detail/edit page

 var name = oraclecrmod.getField("Name");

 if (name != null) name.setColor({"labelBgColor": "Black","labelTextColor":"#ff0000","valueBgColor":
 "#ff0000","valueTextColor":"Black"});

 //get the form on the page for Opportunity
 if(oraclecrmod.ctx.object == "Opportunity")
 {

 //for the opportunity list page
 if (oraclecrmod.ctx.isListPage())
 {

 var oppList = oraclecrmod.getList();
 if(oppList != null)
 {
 oppList.on("display", colorOpptyList);
 }
 } else {
 //for detail/edit pages, etc
 if (oraclecrmod.getForm() != null)
 {

 oraclecrmod.getForm().on("display", colorOppty); //set the custom function to run on display of the form

 }
 var contactList = oraclecrmod.getList("ContactRoleChildList");
 if(contactList != null) {
 contactList.on("display", colorOpptyContactChildList); }
 }
 }
});
```