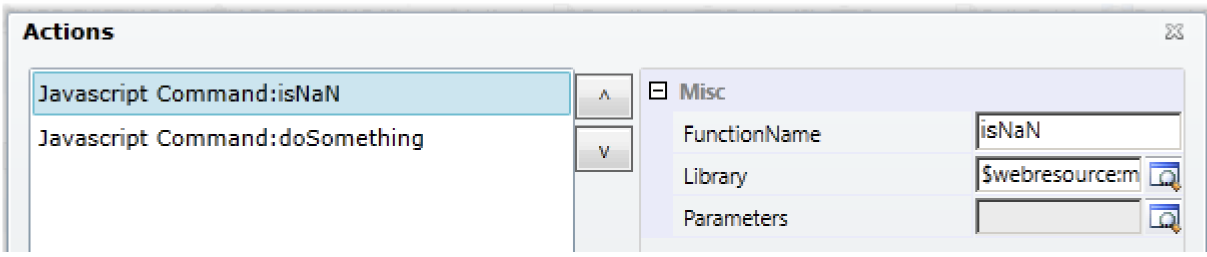
## Contents

## Installation & Usage

1. Contact us to download the version 3.0 trial managed solution, and to discuss pricing. The process of downloading the trial, and purchasing the full version will all be made available online in the near future.
2. Install the managed solution. If updating from the v2.1 unmanaged solution, you will need to delete the solution (not the components) before importing the new managed solution. If you've previously created the alert.js files using your own publisher prefix, it is recommended that you replace these with the 'mag_' files contained in the managed solution for better support going forward, however if you need to keep using your own prefix for now, get in contact with us to get the code.
3. Create a new JavaScript web resource or use an existing web resource to create your custom functions which will call the alert.js functions.

4. Add a reference to the alert.js web resource on your form or view (the other required files are loaded automatically).
    - For forms, this is simply added through the form properties.
    - For views, add a "JavaScript Function Action" to the "Command Actions" for your button command, calling the alert.js web resource, where the function is "isNaN" (without quotes). Using the Ribbon Workbench to edit the command is recommended.

      Javascript Command:isNaN
      Javascript Command:doSomething

      FunctionName: isNaN
      Library: $webresource:m
      Parameters:

    - For Dynamics 365 v9, you can add this reference directly to your Web Resource as a dependency, which will automatically load the Alert.js file from wherever your web resource is used.
5. Add a reference to your custom JavaScript web resource where you're making the calls to alert.js. This should be added after the reference to alert.js.
6. Call your custom function from your JavaScript library, e.g. from the command bar button, or from the form onload event.

## Base Alert object

Create a new instance of an alert, passing in all the options required for the alert. This can then be used and reused with the supported functions, e.g. show(), hide(), etc. Not all options are available for all types of alert. See each individual function for more information about which options are available for each. Options that are not available will be ignored, or may have unexpected results.

```
new Alert(options)
```

### Parameters

#### options

- **Type: Object.** An object containing all the options for displaying this alert. The object contains the following attributes:
    - **title**: String. The main (large) text to display inside the message.

      ```
      "Would you like to create a sale?"
      ```

- **message**: String. The sub-heading (smaller) text to display below the title. This should be no more than a couple of lines of text.

  ```
  "This will create and open the new sale record."
  ```

- **content**: String. This displays directly below the message. This value accepts HTML, and also jQuery elements, so if you want to display more complex messages, you can pass in custom HTML. You can also display plugin traces or error messages here, or even create more complex HTML objects like iframes etc. Note: use the htmlEncode function to display things like XML.

  ```
  "This will create and open the new sale record."
  ```

- **buttons**: Array of Alert.Button objects. The buttons to display inside the alert. If this is set to null, a default "OK" button is displayed, which simply closes the message. If this is set to an empty array, no buttons will be displayed, meaning the alert can only be dismissed using the 'X' or the Alert.hide() function. Note: the default OK button is not displayed when using showIFrame or showWebResource.

  ```
  [
      new Alert.Button("Create Sale", createSaleFunction, true),
      new Alert.Button("Not now")
  ]
  ```

  - Each Alert.Button object should be created using the following properties:
  - `new Alert.Button(label, callback, setFocus, preventClose)`
    - **label (String)**: The text to display on the button.
    - **callback (function)**: A custom function to call when the button is clicked. This function takes a single input parameter, which contains the prompt responses returned by getPromptResponses when using showPrompt, or the iFrame window returned by getIFrameWindow when using showIFrame and showWebResource. This function can be declared inline as a dynamic function. A callback is not required, as the button can still close the alert by default (unless you specify otherwise).
    - **setFocus (bool)**: Specify whether this button should have focus set to it by default, i.e. to allow pressing 'enter' to fire the button command. If there is only one button, focus is automatically set to the button, unless you explicitly set this to false. Buttons with setFocus set to true will display as a 'blue' primary button.

        - **preventClose (bool)**: Specify whether this button should not close the alert after executing the callback function. By default, if this value is not specified, the alert will be automatically closed after clicking the button. Set this to true if you want to keep the alert open, for example if your button does something like the 'download log' for CRM errors, where you want to keep the error open, or if you want to run some validation before closing the alert.

- **icon**: String. The icon to display beside the message.
    - **ERROR**: Message will display an error icon.
    - **WARNING**: Message will display a warning icon.
    - **INFO**: Message will display an info icon.
    - **SUCCESS**: Message will display a green tick (success) icon.
    - **QUESTION**: Message will display a question mark icon.
    - **LOADING**: Message will display a CRM loading spinner icon.
    - **SEARCH**: Message will display a search icon.

    ```
    "QUESTION"
    ```

- **width**: Number. The width of the dialog in pixels. If not specified, this will default to 500px.

    ```
    500
    ```

- **height**: Number. The height of the dialog in pixels. If not specified, this will default to 250px. If the specified height is greater than 250, the message and icon will align to the top of the alert, which is useful for displaying larger messages or trace logs etc. Otherwise, if the height is 250 or less, the message and icon will align in the middle of the alert, which is best used for simple 1-2 line messages. In either case, if the message content exceeds the height of the alert, a scroll bar will be added allowing the user to read the whole message, and copy the content if needed.

    ```
    250
    ```

- **baseUrl**: String. The CRM server base URL. Not required on forms or views. If not specified, it will default to `Xrm.Page.context.getClientUrl()`. Must be specified where Xrm.Page is not available, e.g. from web resources.

    ```
    "http://server/org"
    ```

- **preventCancel**: bool. Specify whether the 'X' to close the alert should be hidden, preventing the user from closing the alert without using one of the specified buttons. By default the 'X' will be displayed, so to hide the 'X', set this to true.

    ```
    true
    ```

- **allowDismiss**: bool. Specify whether to allow the user to dismiss/cancel the alert by clicking outside of the alert. If this is set to true, clicking anywhere outside of the alert will perform a 'hide' on the topmost alert, without firing any button callbacks. This performs similar to clicking the 'X' in the top right corner. By default, clicking outside of the alert will not close the alert, so to enable this for a specific alert, set this to true.

    ```
    true
    ```

- **padding**: Number. Specify custom padding around the popup (in pixels). If not specified, or set to null, this will default to 20px for standard alerts, 30px for prompts, and 0px for iframes and web resources.

    ```
    30
    ```

- **fullscreen**: bool. Setting this to true will cause the alert to expand to 100% of the available screen size. This overrides the width and height properties.

    ```
    true
    ```

- **id**: String. Set a custom ID to allow you to stack multiple alerts without the previous being overwritten.

    ```
    "saleAlert"
    ```

## Adding additional options

Once you've created a new Alert(), you can add additional options at any time, and then show the alert again to use the additional options. Each of the 'options' attributes can also be used as a function on the base Alert object. These should be added before calling .show() etc, and can be chained together. E.g.:

```
new Alert({ title: "Create new sale", message: "This will create the new sale
record." }).show();
```

is the same as:

```
new Alert().title("Create new sale").message("This will create the new sale
record.").show();
```

which is also the same as:

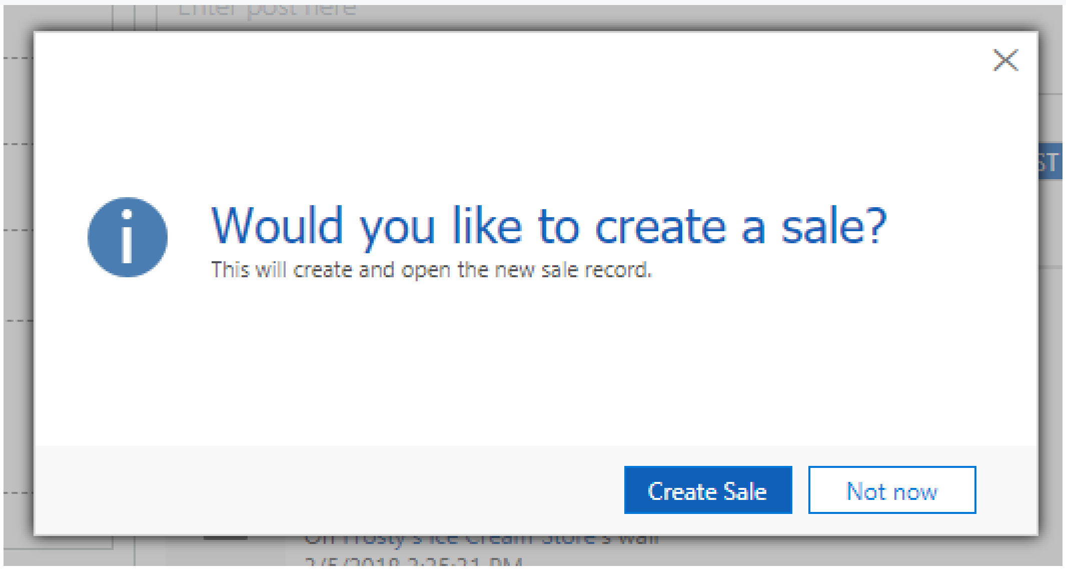
```
var saleAlert = new Alert({ title: "Create new sale"}).show();
saleAlert.message("This will create the new sale record.");
saleAlert.show();
```

This can be useful when creating multi-page prompts, for example, to maintain a base alert object, and modify individual attributes as needed before displaying the next page. All functions (with the exception of the 'get' functions) return the base Alert object, so you can chain them together on one line. This includes the options .title(), .message(), etc, and also the main functions .show(), .hide(), etc.

## show

Use this method to show a light-box message. This can be called from a form, view, or anywhere that supports JavaScript. As of version 2.0, the unsupported CRM light-box is no longer being used. Instead a custom dialog is displayed, 100% contained within the solution, meaning the lightbox displays seamlessly on any browser.

```
new Alert(options).show()
```



Would you like to create a sale?

This will create and open the new sale record.

### Options

See how to create the Base Alert object above for more info on setting these options.

- **title** (defaults to empty string)
- **message** (defaults to empty string)
- **content** (defaults to empty string)
- **buttons** (defaults to a single "OK" button)
- **icon** (defaults to null, i.e. no icon)
- **width** (defaults to 500)

- **height** (defaults to 250)
- **preventCancel** (defaults to false)
- **allowDismiss** (defaults to false)
- **padding** (defaults to 20)
- **fullscreen** (defaults to false)
- **id** (defaults to empty string)

### Example

The following example displays an alert with a custom id, title, and message.

```
var options = { id: "newSale", title: "Create new sale", message: "This will
create a new sale" };
new Alert(options).show();
```

## hide

Use this method to manually hide the alert from code. This is useful if you have a background process running which once completed, will automatically close the alert. This is the default behaviour when closing an alert using the 'X' or by using a button.

```
new Alert(options).hide()
```

### Options

See how to create the Base Alert object above for more info on setting these options.

- **id** (defaults to empty string)

### Example

The following code sample hides an alert with the id "newSale".

```
var options = { id: "newSale" };
new Alert(options).hide();
```

## remove

Use this method to completely remove the alert from the document. By default alerts are only hidden when closed, and will remain in the DOM. Use this function to completely remove it once you're done accessing any information from the alert.

```
new Alert(options).remove()
```

### Options

See how to create the Base Alert object above for more info on setting these options.

- **id** (defaults to empty string)

### Example

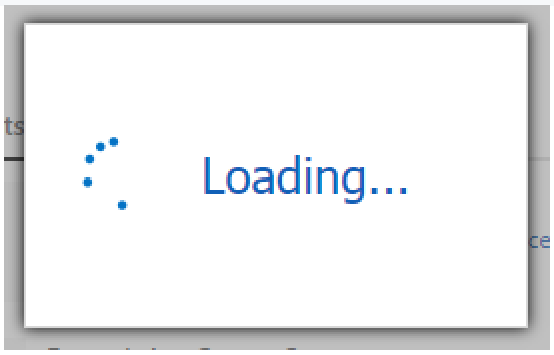
The following code sample removes an alert with the id "newSale".

```
var options = { id: "newSale" };
new Alert(options).remove();
```

## showLoading

Use this method to display a small loading spinner with the text "Loading...". This is just a simple wrapper for the Alert.show method with a few standardised defaults, making it easier to reuse when needing to display a loading message.

```
new Alert(options).showLoading()
```



### Options

See how to create the Base Alert object above for more info on setting these options.

- **title** (defaults to "Loading...")
- **message** (defaults to empty string)
- **buttons** (defaults to [] - i.e. none)
- **icon** (defaults to "LOADING")
- **width** (defaults to 250)
- **height** (defaults to 150)
- **preventCancel** (defaults to true)

- **allowDismiss** (defaults to false)
- **padding** (defaults to 20)
- **id** (defaults to empty string)

### Example

The following example displays a default loading spinner with a custom id.

```
var options = { id: "loading" };
new Alert(options).showLoading();
```

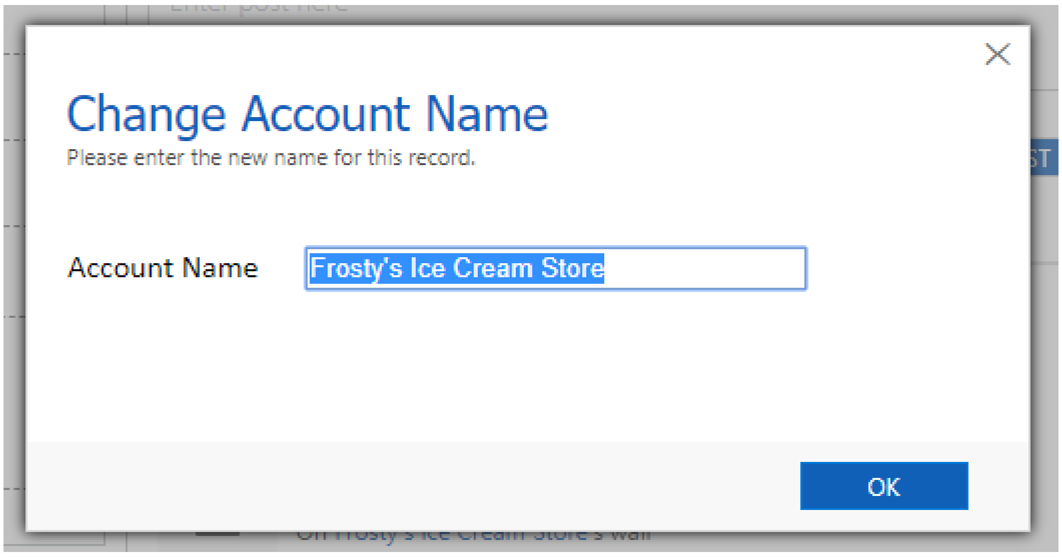
## showWebResource

Use this method to display an HTML web resource inside a light-box. The URL of the web resource is generated and then passed into the Alert.showIFrame function, which creates an iframe to the web resource. You can also optionally add custom titles, buttons, and padding, just like the standard Alert.show function. NOTE: If you're wanting to capture input from a user, consider using showPrompt.

```
new Alert(options).showWebResource(webResourceName)
```

To access the web resource document, i.e. to get the values from inputs on the web resource, you can make a call to getIFrameWindow. This allows you to access any elements inside the web resource being displayed. Note that this is also the first parameter passed to your button callback functions.

To access CRM components from inside the web resource document, i.e. to execute some other form logic on click of a button inside your web resource, you can make a call to getCrmWindow from anywhere inside the web resource. This allows you to call other functions, or access form fields directly from within the web resource.


Change Account Name
Please enter the new name for this record.
Account Name: Frosty's Ice Cream Store

### Options

See how to create the Base Alert object above for more info on setting these options.

- **title** (defaults to empty string)
- **message** (defaults to empty string)
- **buttons** (defaults to [] - i.e. none)
- **width** (defaults to 800)
- **height** (defaults to 600)
- **preventCancel** (defaults to false)
- **allowDismiss** (defaults to false)
- **padding** (defaults to 0)
- **fullscreen** (defaults to false)
- **id** (defaults to empty string)

### Parameters

#### webResourceName

- **Type: String.** The schema name of the HTML web resource to display inside the alert. Additional query string parameters can be added to the webResourceName to pass values to the web resource (must be added to the Data query string).

```
"new_/html/tester.html?Data=" + encodeURIComponent("someVar1=1&someVar2=2")
```

### Example

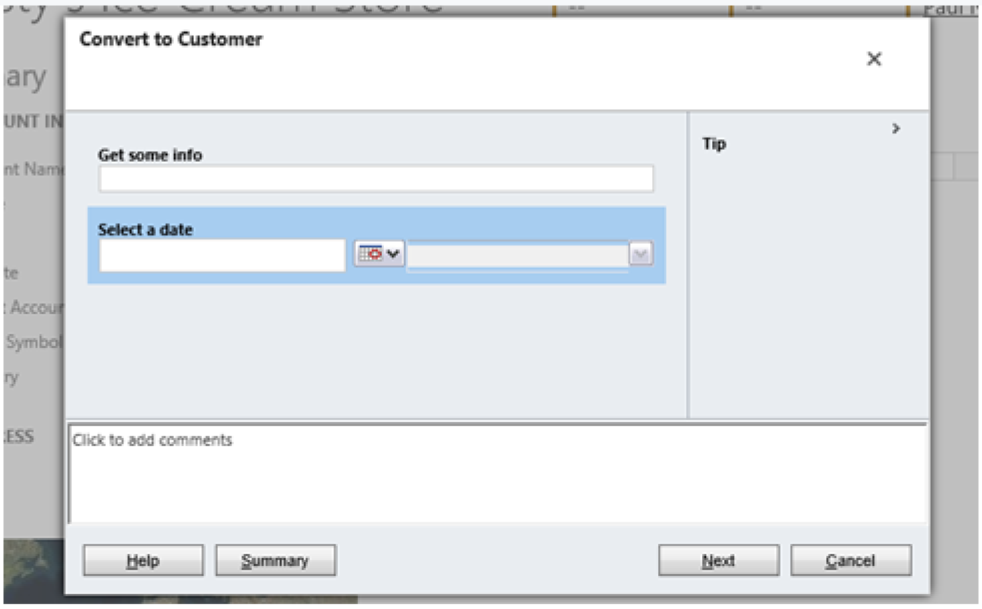
The following example displays a fullscreen web resource, and accesses the value of an input on the web resource in the button callback.

```
var buttonCallback = function(context) {
    alert(context.document.getElementById("someinput").value);
};

var options = { fullscreen: true, buttons: [ new Alert.Button("OK",
buttonCallback) ] };
new Alert(options).showWebResource("new_/html/tester.html");
```

## showDialogProcess

Use this method to display a CRM dialog process inside a light-box. The URL of the dialog process is generated and then passed into the showIFrame function, which creates an iframe to the dialog process. You can also optionally specify a callback function which will execute when the dialog is completed or cancelled, allowing you to refresh the form for example.

```
new Alert(options).showDialogProcess(dialogId, entityName, recordId, callback)
```



## Options

See how to create the Base Alert object above for more info on setting these options.

- **width** (defaults to 800)
- **height** (defaults to 600)
- **preventCancel** (defaults to false)
- **allowDismiss** (defaults to false)
- **padding** (defaults to 0)
- **fullscreen** (defaults to false)
- **id** (defaults to empty string)

## Parameters

### dialogId

- **Type: String.** The Guid of the dialog to display. You can find this by opening the process, and getting the 'id' query string parameter from the URL.

```
"ac03c16e-40b2-41c4-9831-4f651b77f393"
```

### entityName

- **Type: String.** The schema name of the entity the dialog is being run for.

```
"account"
```

**recordId**

- **Type: String.** The Guid of the record to run the dialog against. To run the dialog against the current record (i.e. from a form), you can use `Xrm.Page.data.entity.getId()`.

```
Xrm.Page.data.entity.getId()
```

**callback**

- **Type: function.** A function to call when the dialog is closed. This should include any code you wish to execute after the user is finished with the dialog, e.g. to refresh the form. Note that this function will fire whether the user completed the dialog, or closed it without completing it. The callback function accepts no parameters, and can be declared inline as a dynamic function.

```
function () {
    Xrm.Page.data.refresh();
}
```

## Example

The following example displays a standard dialog process with a basic callback to refresh the form data.

```
var recordId = Xrm.Page.data.entity.getId();
var dialogId = "ac03c16e-40b2-41c4-9831-4f651b77f393";
var callback = function() {
    Xrm.Page.data.refresh();
};

var options = { id: "dialogProcess" };
new Alert(options).showDialogProcess(dialogId, "account", recordId, callback);
```

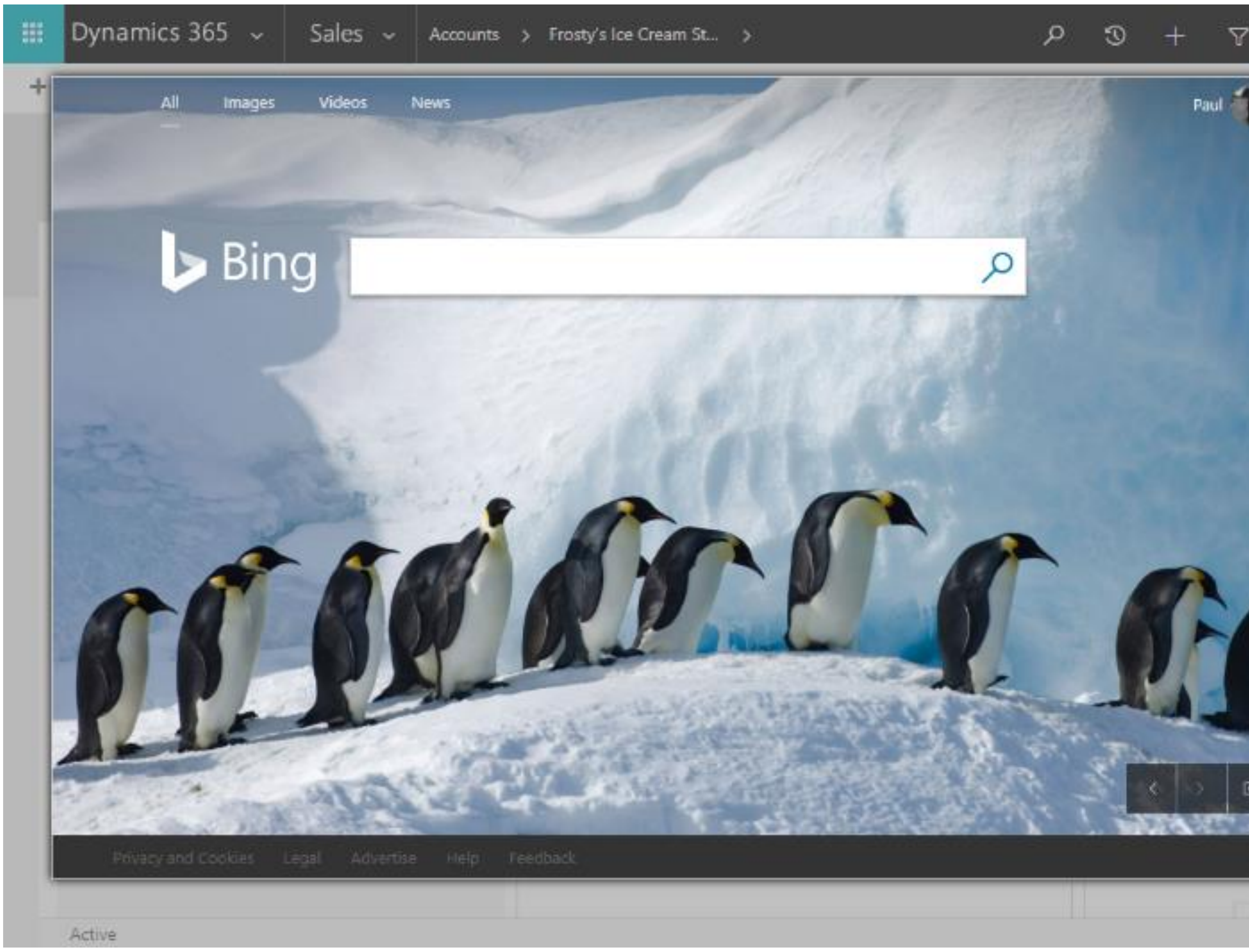
# showIFrame

Use this method to display a webpage or custom HTML inside a light-box, via an iFrame. You can also optionally add custom titles, buttons, and padding, just like the standard show function.

```
new Alert(options).showIFrame(iframeUrl, iframeContent)
```

To access the iframe document, i.e. to get the values from inputs on the web page, you can make a call to getIFrameWindow. This allows you to access any elements inside the web page being displayed. Note that this is also the first parameter passed to your button callback functions. Cross-domain scripting is not allowed, so the web page being displayed in the iframe must be on the same domain as CRM for you to access its document via this method.

To access CRM components from inside the iframe document, i.e. to execute some other form logic on click of a button inside your web page, you can make a call to getCrmWindow from anywhere inside the web page. This allows you to call other functions, or access form fields directly from within the iframe document. Note that cross-domain scripting is not allowed, so the web page being displayed in the iframe must be on the same domain as CRM for you to call into CRM from its document.



## Options

See how to create the Base Alert object above for more info on setting these options.

- **title** (defaults to empty string)
- **message** (defaults to empty string)

- **buttons** (defaults to [] - i.e. none)
- **width** (defaults to 800)
- **height** (defaults to 600)
- **preventCancel** (defaults to false)
- **allowDismiss** (defaults to false)
- **padding** (defaults to 0)
- **fullscreen** (defaults to false)
- **id** (defaults to empty string)

### Parameters

#### iframeUrl

- **Type: String.** The URL of the webpage to display inside the light-box. This webpage must allow being displayed inside an iframe.
  E.g. `"http://bing.com"` works but `"http://google.com"` does not. This parameter can be set to null if iframeContent is being set.

```
"http://bing.com"
```

#### iframeContent

- **Type: String.** Custom HTML to display inside the iframe. Ideally, this should be an entire HTML page, including the body and head tags etc. This will override the iframeUrl if provided. NOTE: This is not supported in IE/Edge.

```
"<p>This is inserted into the iframe document.</p>"
```

### Example

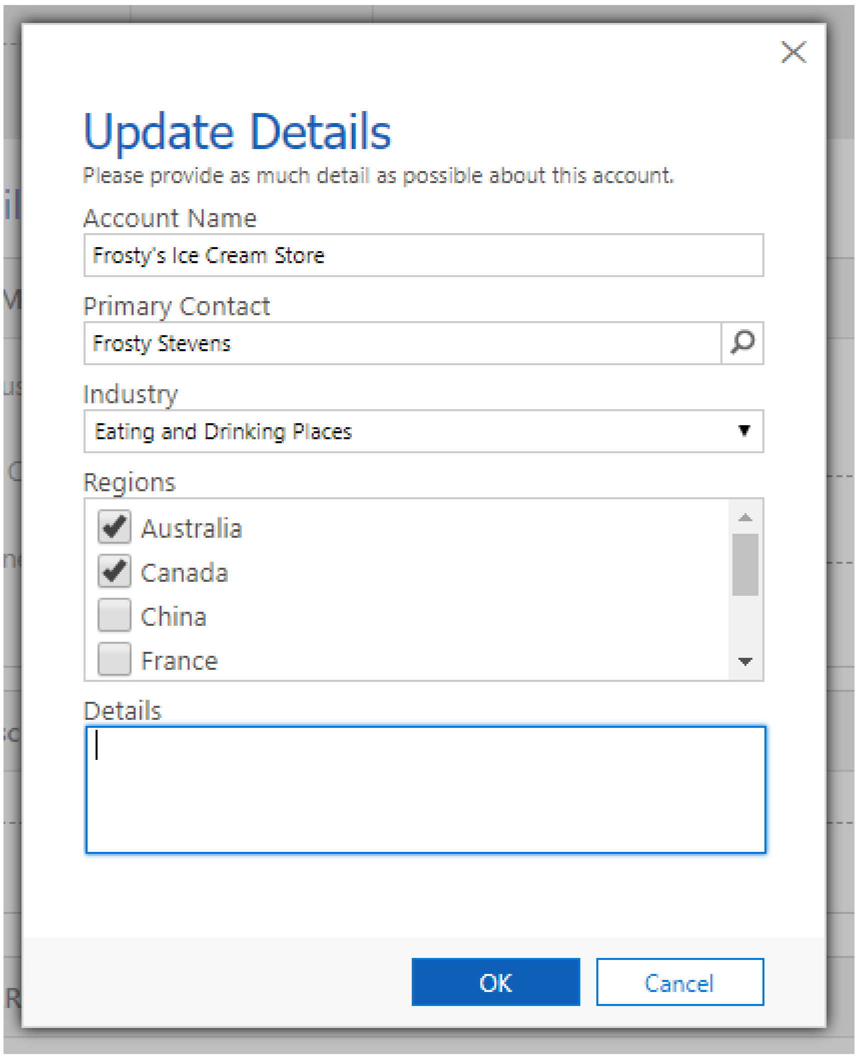
The following example displays a fullscreen iframe to bing.

```
var options = { id: "bingFrame", fullscreen: true };
new Alert(options).showIFrame("https://bing.com");
```

# showPrompt

Use this method to show a prompt containing fields to capture input from the user, then return the responses using getPromptResponses, or using the first parameter inside the button callback, and then use those values to continue processing.

```
new Alert(options).showPrompt(fields)
```


Update Details

Please provide as much detail as possible about this account.

Account Name: Frosty's Ice Cream Store

Primary Contact: Frosty Stevens

Industry: Eating and Drinking Places

Regions: [✓ Australia] [✓ Canada] China France

Details

## Options

See how to create the Base Alert object above for more info on setting these options.

- **title** (defaults to empty string)
- **message** (defaults to empty string)
- **buttons** (defaults to a single "OK" button)
- **icon** (defaults to null, i.e. no icon)
- **width** (defaults to 500)
- **height** (defaults to 250)
- **preventCancel** (defaults to false)
- **allowDismiss** (defaults to false)
- **padding** (defaults to 30)
- **fullscreen** (defaults to false)
- **id** (defaults to empty string)

## Parameters

### fields

- **Type: Array of Objects.** The fields to display inside the prompt. Each type of field is constructed differently.
    - **new Alert.Group(options, extraAttributes)**
        - **options**: Object. The options for the field. The following properties can be set.
            - **id**: String. A unique identifier for the group. This will be used as the 'id' property when getting prompt responses.
            - **label**: String. The label text above the group.
            - **fields**: Array of Objects. The fields to display inside the group. These fields are created exactly as above.
        - **extraAttributes**: Object. Extra HTML attributes that will be added to the div containing the fields. This can include things like: { style: "max-height: 100px" } to add a scrollbar to the group if the fields exceed the max height.
    - **new Alert.Input(options, extraAttributes)**
        - **options**: Object. The options for the field. The following properties can be set.
            - **id**: String. A unique identifier for the field. This will be used as the 'id' property when getting prompt responses.
            - **label**: String. The label text above or beside the field (depending on the type). Checkbox and Radio inputs have the label to the right. If no label is provided, the input will be displayed without a label.
            - **value**: The default value for the input. The type of object depends on the 'type' used for this input. E.g. 'number' takes a Number, 'date' and 'datetime-local' take a Date object, and 'text' takes a String.
            - **type**: String. The type of HTML input to use. This will default to 'text' if not specified. Any HTML input type can be used, but the following are supported:
                - **text**
                - **number**. Specify 'min' and 'max' etc using extraAttributes.
                - **date**. Does not display a date picker in Internet Explorer.
                - **datetime-local**. Does not display a date picker in Internet Explorer.

            - **radio**. Specify 'name' using extraAttributes.
            - **checkbox**
            - **file**. Specify 'multiple' using extraAttributes.
            - **range**
    - **extraAttributes**: Object. Extra HTML attributes that will be added to the input. This can include things like: { name: "radiolist" } to add a name grouping to radio buttons.
- **new Alert.MultiLine(options, extraAttributes)**
    - **options**: Object. The options for the field. The following properties can be set.
        - **id**: String. A unique identifier for the field. This will be used as the 'id' property when getting prompt responses.
        - **label**: String. The label text above the field. If no label is provided, the text area will be displayed without a label.
        - **value**: String. The default value for the text area.
    - **extraAttributes**: Object. Extra HTML attributes that will be added to the text area. This can include things like: { maxlength: 200 } to set a max length on the text area.
- **new Alert.OptionSet(options, extraAttributes)**
    - **options**: Object. The options for the field. The following properties can be set.
        - **id**: String. A unique identifier for the field. This will be used as the 'id' property when getting prompt responses.
        - **label**: String. The label text above the field. If no label is provided, the option set will be displayed without a label.
        - **value**: Number, String, or Array of Strings. The default value for the option set. This should be the 'value' of the option to select by default. If the 'multiple' additionalAttribute is set, this value can be an Array of Strings to select multiple values by default.
        - **options**: Array of Objects. The options to display in the option set. This value uses the exact same structure as the .getOptions() function on a CRM optionset field. Each object in the array consists of the following properties:
            - **text**. String. The text to display in the option set for this option.
            - **value**. Number or String. The value returned if this option is selected. Options with a value of -1 (as a number) or "null" (as a string) will be ignored.

            - **title**. String. The text to display in the tooltip when hovering over the option. If not specified, this will default to the option 'text'.
        - **extraAttributes**: Object. Extra HTML attributes that will be added to the option set. This can include things like: { multiple: "multiple", style: "height: 100px" } to create a multi-select option set with a fixed height of 100px.
    - **new Alert.Lookup(options, extraAttributes)** - Only supported in Dynamics 365 version 9.0 and above
        - **options**: Object. The options for the field. The following properties can be set.
            - **id**: String. A unique identifier for the field. This will be used as the 'id' property when getting prompt responses.
            - **label**: String. The label text above the field. If no label is provided, the lookup will be displayed without a label.
            - **value**: Array of Lookup Objects. The default value for the lookup. This should follow the exact same structure as CRM when using .getValue() from a lookup field. The lookup object should contain 'id', 'name', and 'entityType'. Although this is an Array, only one lookup object is supported.
            - **entityTypes**: Array of Strings. The entity types to look up. E.g. ["contact"] for a contact lookup, or ["account", "contact"] for a customer lookup.
            - **customFilters**: Array of Strings. Custom filters to apply to the lookup results. Each custom filter will be applied to each of the specified entityTypes. These should **not**be URL encoded.
        - **extraAttributes**: Object. Extra HTML attributes that will be added to the lookup text box.

```
[new Alert.Input({ label: "Name" }), new Alert.MultiLine({ label: "Details" })]
```

## Example

The following example displays a prompt with 2 fields, and accesses the values in the button callback.

```
var options = {
    id: "promptExample",
    buttons: [new Alert.Button("OK", function(results) {
        var subscribe = results[0].value;
        var details = results[1].value;
        alert("Subscribe: " + subscribe);
        alert("Details: " + details);
    })]
};

var alertBase = new Alert(options);
alertBase.showPrompt([
    new Alert.Input({ label: "Subscribe", type: "checkbox" }),
    new Alert.MultiLine({ label: "Details" })
]);
```

## getIFrameWindow

Use this method to get the context of an iFrame being displayed in the light-box. For example, if you're capturing input via a web resource, you can use this function to access the inputs on the web resource from inside a custom function. This allows you to use custom buttons to access the iFrame data. This value is also returned as the first parameter of each button callback when using showIFrame or showWebResource.

```
new Alert(options).getIFrameWindow()
```

### Options

See how to create the Base Alert object above for more info on setting these options.

- **id** (defaults to empty string)

### Return type

- **Type: window object.** The window representing the current iFrame being displayed in the light-box. If no iFrames are available, this will return null.

### Example

The following example displays a web resource, and gets the iframe window in the button callback.

```
var options = { id: "webResource" };
var alertBase = new Alert(options);

var buttonCallback = function (context) {
    var iframeWindow = alertBase.getIFrameWindow(); // This is the same as the
context parameter
    alert(iframeWindow.document.getElementById("somefield").value);
};

alertBase.buttons([new Alert.Button("OK", buttonCallback)]);
alertBase.showWebResource("new_/html/tester.html");
```

## getCrmWindow

This method gives you context of the CRM form (or client API wrapper if turbo forms are enabled). This allows you to access CRM functions, and your own custom JavaScript libraries from inside iFrame alerts. From inside a web resource, for example, you can call `new parent.Alert().getCrmWindow().doSomething();`, where "doSomething()" represents a custom function loaded onto the parent form.

```
new Alert().getCrmWindow()
```

### Return type

- **Type: window object.** The window representing the CRM form, or client API wrapper if turbo forms are enabled. This window contains all of your custom web resource libraries loaded onto the form.

### Example

```
var crmWindow = new parent.Alert().getCrmWindow();

crmWindow.Xrm.Page.getAttribute("name").setValue("something")
```

## getPromptResponses

This method gets the responses from the last displayed prompt. Responses are returned in the same order they are displayed. This value is also returned as the first parameter of each button callback when using showPrompt.

```
new Alert(options).getPromptResponses()
```

### Options

See how to create the Base Alert object above for more info on setting these options.

- **id** (defaults to empty string)

### Return type

- **Type: Array of Objects.** The responses from all fields on a prompt. Each object consists of an 'id' and a 'value'. Each response is returned in the same order as it is added to the prompt, so they can be accessed by their index in the array or by checking their 'id' attribute.

- **id (String)**: The 'id' exactly matches the 'id' given to the field when creating the prompt. If no 'id' was specified on the field, this will be an empty string.
- **value (Object)**: The 'value' represents the field value. This is different depending on the type of field used. Empty values will become 'null'.
  - **Alert.Group**: Returns an Array of Objects. This contains all the fields within this group. Each object follows the same structure as above, consisting of an 'id' and 'value'.
  - **Alert.Input**: The value returned will depend on the HTML input 'type' used. The following values will be returned for different HTML input types.
    - **text**: Returns a String.
    - **date**: Returns a Date.
    - **datetime-local**: Returns a Date.
    - **radio**: Returns a bool. Returns true if checked, otherwise false.
    - **checkbox**: Returns a bool. Returns true if checked, otherwise false.
    - **number**: Returns a Number.
    - **range**: Returns a Number.
    - **file**: Returns an Array of File Objects. Returns the data for the uploaded file(s). Each file object has the following attributes:
      - **lastModifiedDate**: Date. The last modified date of the file.
      - **name**: String. The file name.
      - **size**: Number. The file size in bytes.
      - **type**: String. The file type, e.g. "text/plain".
      - **value**: String. The file contents in base64.
  - **Alert.OptionSet**: Returns a Number, String, or Array of Strings. This is the value from the selected option(s). If this is numeric, it will return a Number, otherwise a String. If the 'multiple' attribute is added to the field, this will return an Array of the selected options as Strings.
  - **Alert.Lookup**: Returns an Array of Lookup Objects. This returns the same structure as a CRM lookup using .getValue(). The lookup object consists of a 'name', 'id', and 'entityType'.
  - **Alert.MultiLine**: Returns a String.

### Example

The following example displays a prompt with 2 fields, and accesses the values in the button callback.

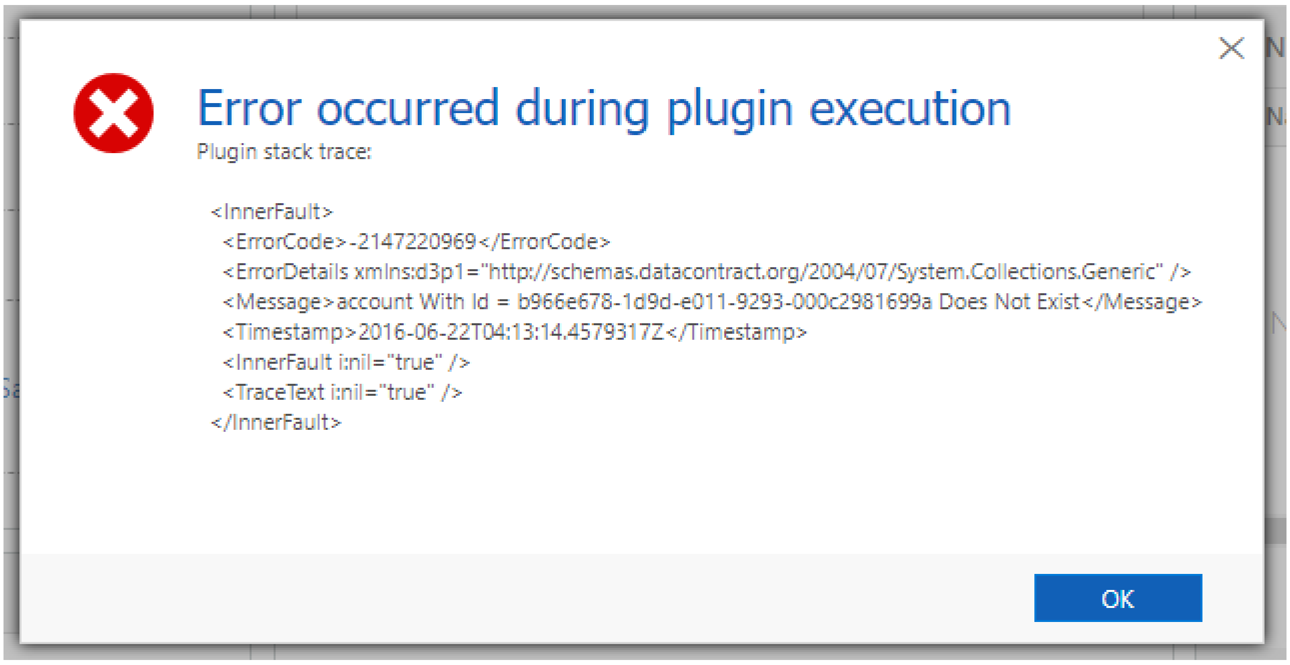
```
var alertBase;
var options = {
    id: "promptExample",
    buttons: [new Alert.Button("OK", function(results) {
        var responses = alertBase.getPromptResponses(); // This is exactly the
same as 'results'

        var subscribe = responses[0].value;
        var details = responses[1].value;

        alert("Subscribe: " + subscribe);
        alert("Details: " + details);
    })]
};

alertBase = new Alert(options);
alertBase.showPrompt([
    new Alert.Input({ label: "Subscribe", type: "checkbox" }),
    new Alert.MultiLine({ label: "Details" })
]);
```

## htmlEncode

Use this method to encode a custom message which contains HTML characters, to allow it to be displayed inside the alert message. For example, if displaying formatted XML with indented spacing and XML tags, calling this method will format the text into an HTML friendly message that displays nicely inside the alert. The returned value should then be passed to the 'message' of the Alert.show method.

NOTE: If you want to actually use HTML tags, like **<b>bold</b>**, you should not use this method.

```
new Alert().htmlEncode(text)
```



```
<InnerFault>
  <ErrorCode>-2147220969</ErrorCode>
  <ErrorDetails xmlns:d3p1="http://schemas.datacontract.org/2004/07/System.Collections.Generic" />
  <Message>account With Id = b966e678-1d9d-e011-9293-000c2981699a Does Not Exist</Message>
  <Timestamp>2016-06-22T04:13:14.4579317Z</Timestamp>
  <InnerFault i:nil="true" />
  <TraceText i:nil="true" />
</InnerFault>
```

## Parameters

### text

- **Type: String.** The text to encode, e.g. a trace log.

```
"  <InnerFault>\n" +
"    <ErrorCode>-2147220969</ErrorCode>\n" +
"    <ErrorDetails
xmlns:d3p1=\"http://schemas.datacontract.org/2004/07/System.Collections.Generic\"
/>\n" +
"    <Message>account With Id = b966e678-1d9d-e011-9293-000c2981699a Does Not
Exist</Message>\n" +
"    <Timestamp>2016-06-22T04:13:14.4579317Z</Timestamp>\n" +
"    <InnerFault i:nil=\"true\" />\n" +
"    <TraceText i:nil=\"true\" />\n" +
"  </InnerFault>"
```

## Return type

- **Type: String.** The text passed into the method, with all HTML tags encoded, including line breaks and spacing.

```
"&nbsp;&nbsp;&lt;InnerFault&gt;<br />&nbsp;&nbsp;&nbsp;&nbsp;&lt;ErrorCode&gt;-
2147220969&lt;/ErrorCode&gt;<br />&nbsp;&nbsp;&nbsp;&nbsp;&lt;ErrorDetails
xmlns:d3p1=&quot;http://schemas.datacontract.org/2004/07/System.Collections.Generi
c&quot; /&gt;<br />&nbsp;&nbsp;&nbsp;&nbsp;&lt;Message&gt;account With Id =
b966e678-1d9d-e011-9293-000c2981699a Does Not Exist&lt;/Message&gt;<br
/>&nbsp;&nbsp;&nbsp;&nbsp;&lt;Timestamp&gt;2016-06-
22T04:13:14.4579317Z&lt;/Timestamp&gt;<br />&nbsp;&nbsp;&nbsp;&nbsp;&lt;InnerFault
i:nil=&quot;true&quot; /&gt;<br />&nbsp;&nbsp;&nbsp;&nbsp;&lt;TraceText
i:nil=&quot;true&quot; /&gt;<br />&nbsp;&nbsp;&lt;/InnerFault&gt;"
```

### Example

The following example encodes a plugin trace log containing XML markup, line breaks, and spacing, and then displays it in the message of an alert.

```
var errorTrace =
    "  <InnerFault>\n" +
    "    <ErrorCode>-2147220969</ErrorCode>\n" +
    "    <ErrorDetails
xmlns:d3p1=\"http://schemas.datacontract.org/2004/07/System.Collections.Generic\"
/>\n" +
    "    <Message>account With Id = b966e678-1d9d-e011-9293-000c2981699a Does Not
Exist</Message>\n" +
    "    <Timestamp>2016-06-22T04:13:14.4579317Z</Timestamp>\n" +
    "    <InnerFault i:nil=\"true\" />\n" +
    "    <TraceText i:nil=\"true\" />\n" +
    "  </InnerFault>";

var options = {
    id: "pluginError",
    title: "Error occurred during plugin execution",
    message: new Alert().htmlEncode("Plugin stack trace:\n\n" + errorTrace),
    icon:"ERROR",
    width: 620,
    height: 310
};

new Alert(options).show();
```

# Alert.$

This method acts as a wrapper for CRM's jQuery function, which can be used to access elements from an Alert. The context can be set to a particular alert using the .get() function. If the context is not provided, this will default to the top Alert context, which will apply to all alerts being displayed at the time.

```
Alert.$(selector, context)
```

## Parameters

#### selector

- **Type: String.** The jQuery selector expression to use. This works exactly the same as the selectors for jQuery normally.

```
"#someInput"
```

#### context

- **Type: Element or jQuery object.** An optional parameter to filter the selector to within a certain context, i.e. a particular alert. This works exactly the same as the second context parameter for jQuery normally.

```
new Alert({ id: "someAlert" }).get()
```

### Return type

- **Type: jQuery object.** The jQuery object containing the results of the selector query. This is exactly the same as what jQuery normally returns.

### Example

The following example gets the jQuery object for an alert, and uses it to access the content of the alert to add an onchange event to one of the inputs.

```
var options = { id: "promptExample" };
var alertBase = new Alert(options);
alertBase.showPrompt([new Alert.Input({ id: "inputExample" })]);

var $alert = alertBase.get();
var $inputExample = Alert.$("#inputExample input", $alert);
$inputExample.on("change", function() { alert(this.value); });
```

## get

This method gets the jquery object for this alert - used for accessing specific inner elements when multiple alerts are displayed. This must be called after showing the alert.

```
new Alert(options).get()
```

### Options

See how to create the Base Alert object above for more info on setting these options.

- **id** (defaults to empty string)

### Return type

- **Type: jQuery object.** The jQuery object wrapper for a particular alert being displayed.

### Example

The following example gets the jQuery object for an alert, and uses it to access the content of the alert to add an onchange event to one of the inputs.

```
var options = { id: "promptExample" };
var alertBase = new Alert(options);
alertBase.showPrompt([new Alert.Input({ id: "inputExample" })]);

var $alert = alertBase.get();
var $inputExample = Alert.$("#inputExample input", $alert);
$inputExample.on("change", function() { alert(this.value); });
```

## Deprecated functions

Version 3.0 has deprecated the old way of creating alerts, e.g. using `Alert.show("Create sale", "This will create a sale");`. These functions are still included in the solution, and will continue to work as they did in v2.1. Old code should be updated to follow the new structure going forward.