Duplicate Detection Sample Read Me

This document describes what this sample does, the contents of this solution, how you can verify that it works, and how you can re-use components in this sample to apply similar duplicate detection for other entities and forms.

**Important**

Do not install this managed solution on your production environment. It includes modifications to the Lead and Contact system entity forms. These changes could disrupt people’s use of Microsoft Dynamics CRM. Install this on a test environment to observe how it works.

Contents

[Goal 1](#_Toc365965454)

[Try it 1](#_Toc365965455)

[Contents 2](#_Toc365965456)

[How to use the components in this sample 4](#_Toc365965457)

[Form Libraries 4](#_Toc365965458)

[OnSave Event Handler 4](#_Toc365965459)

[Apply this pattern to use for another entity form 5](#_Toc365965460)

[How this sample works 7](#_Toc365965461)

# Goal

Client side notification of potential duplicate records was not included in the initial release of Microsoft Dynamics CRM 2013. This sample is intended to provide an example of how this functionality can be added by using client scripts and web resources.

# Try it

When this managed solution is installed you can try it right away using the **Contact** or **Lead** entities. Both of these entities have custom forms that use a JavaScript web resource configured to the **OnSave** event of the form. In order to see the behavior you must use one of these custom forms, so you might have to switch main forms or modify the form order for quick create forms before you can see the behavior.

|  |  |  |
| --- | --- | --- |
| Entity | Form Name | Form Type |
| Contact | Contact form for Duplicate Detection Sample | Main |
|  | Contact Quick Create with Duplicate Detection | Quick Create |
| Lead | Lead form for Duplicate Detection Sample | Main |
|  | Lead Quick Create for Duplicate Detection | Quick Create |
| Duplicate Detect Test | Information | Main |

### Existing duplicate detection rules

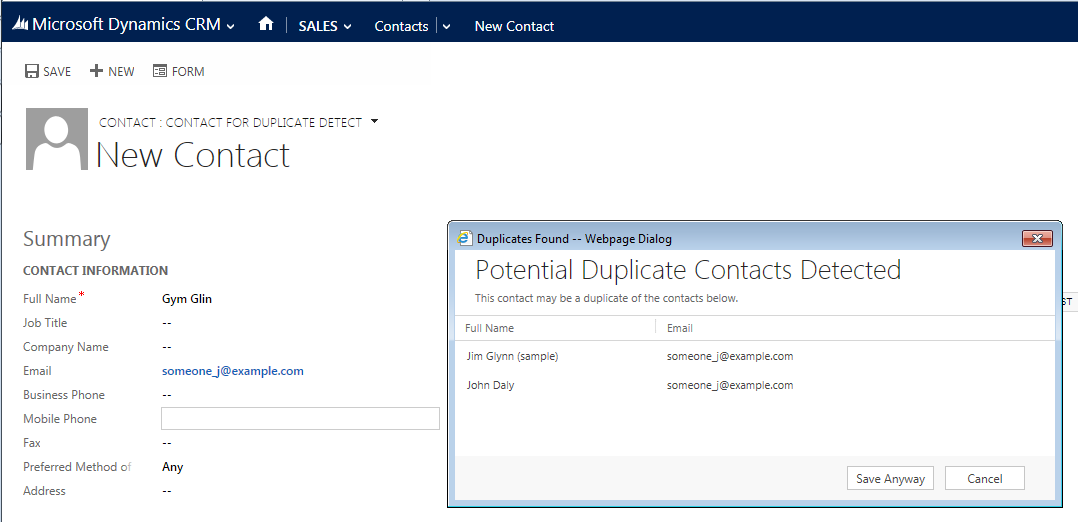
These forms assume that the duplicate detection rules in place for this entity are the default duplicate detection rules. By default the **Contact** and **Lead** entities have the following published duplicate detection rules:

|  |  |
| --- | --- |
| Entity | Rule Name |
| Contact | Contacts with the same last name |
| Contacts with the same business phone number |
| Contacts with the same e-mail address |
| Lead | Leads with the same e-mail address |

The **Duplicate Detect Test** custom entity can be found in the **Settings** area. The main purpose of this entity is to provide another example using a variety of different custom attributes of different types. It is not enabled for quick create.

As a custom entity, **Duplicate Detect Test** does not have any duplicate detection rules. To use this entity to test the sample code you must first create and publish a new duplicate detection rule. For example, use a rule to determine if the **Name** (sample\_name) field is an exact match.

When you attempt to save a new record the data in the record is sent to the server to be compared against existing records. If any matching records are found, the save will be prevented and you will see a modal dialog like this:

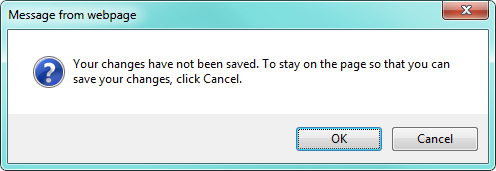


At this point you have three options:

* **Open Existing**: Click a row in the dialog to open the existing record in a new window.
* **Cancel**: Click **Cancel** to close the dialog without saving the record.
* **Save Anyway**: Click **Save Anyway** to save the new record.

### Open Existing

If you choose to open an existing record, the record you were creating will remain unsaved in the original window. When you navigate away from the form in the original window you will be prompted to stay on the page or continue to navigate away without saving. Click **OK** to navigate away.



### Cancel

If you choose **Cancel** the record is not saved and the dialog will appear again if you attempt to save the record.

### Save Anyway

If you choose **Save Anyway**, you can continue to edit and save the record without the dialog appearing again.

## Update Not Enabled

The **OnSave** event code is also executed when you update existing records. The code in this sample is only intended to detect duplicates when new records are being created.

# Contents

This solution contains the following major components:

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| SDK.DuplicateDetection.js | JavaScript Web Resource | This library provides the core components of this sample. It contains the **SDK.Sample.DuplicateDetection.onSave** function that is set to the **OnSave** event handler. You do not need to edit the contents of this library. |
| Duplicate Detect Test | Custom Entity | This custom entity includes a number of different fields used to verify the functionality of this sample. The main form is configured with an example in the main form |
| Account | System entity | The **Account** entity is included because the **Duplicate Detect Test** entity has a lookup to the account entity. It is not configured for duplicate detection |
| Contact | System entity | The **Contact** entity has two forms with **OnSave** event handlers configured to detect duplicate records. |
| Lead | System entity | The **Lead** entity has two forms with OnSave event handlers configured to detect duplicate records. |
| ShowDuplicateContacts.html | HTML Web Resource | This HTML web resource is launched as a modal dialog when a duplicate record is detected on a save event for the **Contact** entity |
| ShowDuplicateContacts.js | JavaScript Web Resource | This JavaScript web resource provides the logic to show duplicate records for **ShowDuplicateContacts.html**. |
| ShowDuplicateLeads.html | HTML Web Resource | This HTML web resource is launched as a modal dialog when a duplicate record is detected on a save event for the **Lead** entity |
| ShowDuplicateLeads.js | JavaScript Web Resource | This JavaScript web resource provides the logic to show duplicate records for **ShowDuplicateLeads.html.** |
| ShowDuplicateTests.html | HTML Web Resource | This HTML web resource is launched as a modal dialog when a duplicate record is detected on a save event for the **Duplicate Detect Test** entity. |
| ShowDuplicateTests.js | JavaScript Web Resource | This JavaScript web resource provides the logic to show duplicate records for **ShowDuplicateTests.html.** |
| jquery\_1.9.1.min.js | JavaScript Web Resource | Each of the **ShowDuplicate**\* web resources depend on jQuery. |
| ShowDuplicates.css | CSS Web Resource | Each of the **ShowDuplicate**\* html web resources includes a link to this style sheet. |

# How to use the components in this sample

To understand how you might use components included in this sample you need to understand how they work together. This section will explain how the forms included within this solution are configured. To implement this duplicate detection functionality for other forms you need to follow the pattern described here.

## Form Libraries

Each of the forms that are configured for duplicate detection have the **SDK.DuplicateDetection.js** (sample\_/duplicateDetection/scripts/SDK.DuplicateDetection.js) library added to the form.

## OnSave Event Handler

The **SDK.Sample.DuplicateDetection.onSave** function from the **SDK.DuplicateDetection.js** library is configured for the **OnSave** event.

Each **OnSave** event is configured to with the **Pass execution context as first parameter** option selected. This is required.

Each **OnSave** event has certain configuration set within the **Comma separated list of parameters that will be passed to the function** area. Each of these parameters is passed to the **SDK.Sample.DuplicateDetection.onSave** function after the execution context. The following table describes what they do:

|  |  |
| --- | --- |
| Parameter | Description |
| disableAutoSave | **Boolean** - Whether the auto-save timer should be disabled. |
| matchingEntityName | **String** - The logical name of the entity to check for duplicates. In each of the examples it is the same entity, but it is possible to check for other entities. For example **Leads** that represent **Contacts** already in the system. |
| primaryIdAttribute | **String** - The logical name of the **PrimaryId** attribute for the entity set by the **matchingEntityName** parameter. |
| attributeParams | **Array** - An array of attribute logical names. These should include the primary name attribute for the entity defined by the **matchingEntityName** parameter. Any attributes used in duplicate detection rules should also be included as well as any attributes that you want to display as columns in the list of duplicate records. |
| webResource | **String** - the Name of the HTML web resource to launch when duplicates are detected. |
| modalDialogArgs | **String** - The arguments parameter passed to the [window.showModalDialog](http://msdn.microsoft.com/en-us/library/ie/ms536759(v=vs.85).aspx) method used to open the web resource specified in the **webResource** parameter. You may use this to set the height and width of the dialog along with any of the other available parameters.  **Note**: Mozilla Firefox ignores the **center** parameter option so the code calculates a center position based on **dialogHeight** and **dialogWidth.** These two parameters are required. The parameters **dialogTop** and **dialogLeft** are used to set the position. |

The following shows the configurations used for entity forms in this solution:

For the **Contact** entity forms:

false,

"contact",

"contactid",

["fullname", "firstname", "lastname","emailaddress1"],

"sample\_/duplicateDetection/ShowDuplicateContacts.html",

"dialogHeight: 250px; dialogWidth: 600px; resizable: yes; status: no;"

For the **Lead** entity forms:

false,

"lead",

"leadid",

["subject","fullname", "firstname", "lastname","emailaddress1","address1\_city"],

"sample\_/duplicateDetection/ShowDuplicateLeads.html",

"dialogHeight: 250px; dialogWidth: 650px; resizable: yes; status: no;"

For the **Duplicate Detect Test** entity forms:

false,

"sample\_duplicatedetecttest",

"sample\_duplicatedetecttestid",

["sample\_name","sample\_account","sample\_bool","sample\_date","sample\_decimal","sample\_double","sample\_integer","sample\_memo",

"sample\_money","sample\_optionset"],

"sample\_/duplicateDetection/ShowDuplicateTests.html",

"dialogHeight: 250px; dialogWidth: 800px; resizable: yes; status: no;"

# Apply this pattern to use for another entity form

The following provide general instructions about how to make this duplicate detection notification solution work for other entities and forms.

1. Copy one of the existing **ShowDuplicate\*.html** pages from the Visual Studio solution and give it a name consistent with the other html pages.
2. Copy the **ShowDuplicate\*.js** files and rename it. Edit your **ShowDuplicate\*.html** web resource so that it uses this file.
3. The **ShowDuplicate\*.html** pages are the content of the modal dialog. They include the text and the headings of the grid displaying the duplicate records found. Edit this file to include the appropriate text and grid headings. You will find a table **colgroup** that specifies the column widths and a single row that specifies the headings.
4. The **ShowDuplicate\*.js** files accept an array of duplicate records in the dialog arguments. The fields for the records depend on the attribute names you set in the **attributeParams** parameter in the configuration of the **OnSave** event. Within the **showRecords** function you will find the code to define the columns widths and write rows for the table.
5. Because the grid does not automatically match the heading in the HTML web resource, you need to adjust the columns to be the same width. The number of columns can be less than the number of attributes defined in the **attributeParams** parameter, but all the attributes to be displayed must be included in the **attributeParams** parameter. The code that specifies the columns looks like this:

var colGroup = $("<colgroup />");

colGroup.append($("<col width='50' class='sdk-sample-List-DataColumn' name='sample\_name' />"));

colGroup.append($("<col width='100' class='sdk-sample-List-DataColumn' name='sample\_account' />"));

colGroup.append($("<col width='50' class='sdk-sample-List-DataColumn' name='sample\_bool' />"));

colGroup.append($("<col width='50' class='sdk-sample-List-DataColumn' name='sample\_date' />"));

colGroup.append($("<col width='50' class='sdk-sample-List-DataColumn' name='sample\_decimal' />"));

colGroup.append($("<col width='50' class='sdk-sample-List-DataColumn' name='sample\_double' />"));

colGroup.append($("<col width='50' class='sdk-sample-List-DataColumn' name='sample\_integer' />"));

colGroup.append($("<col width='50' class='sdk-sample-List-DataColumn' name='sample\_memo' />"));

colGroup.append($("<col width='50' class='sdk-sample-List-DataColumn' name='sample\_money' />"));

colGroup.append($("<col width='50' class='sdk-sample-List-DataColumn' name='sample\_optionset' />"));

colGroup.append($("<col />"));

resultsTable.append(colGroup);

1. You will find code to write the rows of the table. The first column in the row should be the primary name attribute for the entity. It contains a link, so it can be opened with a click, but the entire row can be opened with a double click.

//primary\_attribute\_name column

var primary\_attribute\_nameCell = $("<td class='sdk-sample-List-DataCell inner-grid-cellPadding' />");

var primary\_attribute\_nameGridcellpadding = $("<nobr class='gridcellpadding' />");

var primary\_attribute\_nameLink = $("<a class='sdk-sample-List-Link' />");

primary\_attribute\_nameLink.attr("title", record.primary\_attribute\_name)

primary\_attribute\_nameLink.attr("tabindex", i)

primary\_attribute\_nameLink.attr("href", "#")

primary\_attribute\_nameLink.attr("target", "\_blank")

primary\_attribute\_nameLink.attr("oid", record.id)

primary\_attribute\_nameLink.attr("otypename", 'sample\_duplicatedetecttest')

primary\_attribute\_nameLink.text(record.primary\_attribute\_name);

primary\_attribute\_nameGridcellpadding.append(primary\_attribute\_nameLink);

primary\_attribute\_nameCell.append(primary\_attribute\_nameGridcellpadding);

primary\_attribute\_nameCell.append(primary\_attribute\_nameGridcellpadding);

row.append(primary\_attribute\_nameCell);

primary\_attribute\_nameLink.on("click", openRecord);

1. The remaining columns follow a consistent pattern like so:

//attributeName column

var attributeNameCell = $("<td class='sdk-sample-List-DataCell inner-grid-cellPadding' />");

var attributeNameGridcellpadding = $("<nobr class='gridcellpadding' />");

attributeNameGridcellpadding.attr("title", record.attributeName)

attributeNameGridcellpadding.text(record.attributeName);

attributeNameCell.append(attributeNameGridcellpadding);

row.append(attributeNameCell);

You will probably be able to use find/replace within each column to give each variable a unique name based on the name of the attribute you want to display.

1. After you have edited the HTML and JS files, create web resources using them, and then configure event handlers for the form as described in [How to use the components in this sample](#_How_to_use).

# How this sample works

This section will walk through the code found in the **SDK.DuplicateDetection.js** and explain the logic used in the sample.

**Note:**

Unless otherwise stated, each object uses **SDK.DuplicateDetection** as a namespace.

When a save occurs the **onSave** function executes.

If the **disableAutoSave** configuration parameter is true, the automatic save that occurs after 30 seconds is disabled.

After some code verifies that all the expected parameters are present and of the correct type, the type of form is checked. Duplicate detection logic is only applied to the create form.

The **saveMode** is captured in case the save is allowed. Only ‘saveandclose’ and ‘saveandnew’ modes can be used, so if those were the **saveMode**, they will be applied if the save is allowed.

There is a global variable called **allowSave**. By default this is false so that save can be prevented until it is determined that the record is not a duplicate. If **allowSave** is true the save will occur uninterrupted. If it is false, the save is prevented and the attributes passed in the **attributeParams** parameter are processed using an **Attribute** function. This function serves as a class that captures the value and type of the attribute. It has an override to the **toString** function to generate the appropriate XML when they are used in a SOAP request.

The array of attributes and other parameters are passed to the constructor of a **Request** function that serves as a class to prepare to execute a **RetrieveDuplicatesRequest** message using the SOAP endpoint.

This message allows for setting pages to retrieve paged data, but this capability is not used in this sample. Only the first 50 detected duplicates will be returned. This sample has no capability to retrieve additional pages.

The **Request** class has an **Execute** method that leverages an **SDK.Sample.SOAP.Execute** class to perform the actual request. This class accepts the **requestXML** as well as callback functions and a pointer to a class that can be used to process the response. In this case the **Response** class is responsible for parsing the XML returned and providing a single public method (**getDuplicateRecords**) to receive the data as an array JavaScript objects. The **Response** class contains a number of internal functions used to process the XML returned. The internal **\_objectifyRecord** function leverages the attribute definitions that were passed through to generate JavaScript objects with properties that can be displayed in the grid. Rather than setting each attribute to the matching type, this function leverages the data in the **FormattedValues** collection to set string values for types like optionSets where the data is an integer but a corresponding label is included in the **FormattedValues** collection.

The **onSave** function is complete when it uses the **Request.Execute** method and specifies that **manageDuplicates** is the **successCallback** and will accept the **Response** instance passed back.

The **manageDuplicates** function uses the **Response**.**getDuplicateRecords** function to examine how many duplicate records were found. If no duplicates are found, **saveRecord** is called to perform the save.

If there are duplicates, the HTML web resource is opened using window.[showModalDialog](http://msdn.microsoft.com/en-us/library/ie/ms536759(v=vs.85).aspx) and the duplicate records are passed through in the [dialogArguments](http://msdn.microsoft.com/en-us/library/ie/ms533723(v=vs.85).aspx) parameter.

At this point, the code is blocked awaiting a response from the dialog. This code cannot work for the **CRM for Tablets** client, so it will only be called if Xrm.Page.context.client.getClient() != “Mobile”.

At this point, the **ShowDuplicate\*.html** web resource has only to show the duplicate records and process the result. If the user clicks **Save Anyway**, the record will be saved. Any other outcome, including opening one of the duplicate records, leaves the record unsaved. When the user attempts to close the window or navigate to another record, they will be notified that their changes are not saved. They can choose to click **OK** and allow the new record to not be created or for the change to an existing record to not be saved.