Configuration Guide

Dynamics CRM Connect

A guide to configuring the Dynamics CRM connector for Sitecore Experience Platform

sitecore®
Own the experience™

Table of Contents

Chapter 1

Getting Started

This chapter guides you through the initial configuration for Dynamics CRM Connect.

1.1 Prerequisites

This section assumes you have:

- Installed Dynamics CRM Connect 1.0 on your Sitecore server
- A Dynamics CRM system

Note
If you do not have access to a Dynamics CRM server, you can create a free 30-day trial account on Dynamics CRM Online. More information is available at https://www.microsoft.com/en-us/dynamics/crm-free-trial-overview.aspx.

1.2 Initial Configuration

After you have installed Dynamics CRM Connect, you configure Sitecore to connect to your CRM.

1.2.1 Add a new Tenant

All of the settings that pertain to synchronizing your Sitecore server with your CRM are grouped under an item called a Tenant.

You must create a Tenant in order to configure a connection to your CRM.

1. In Sitecore, open Content Editor.
2. Navigate to `/sitecore/system/Data Exchange`.
3. Add the following item:
    a. **Template**: Data Exchange Tenant for Dynamics CRM
    b. **Name**: mycrm (You can use any value you want. This value should describe the CRM server you will connect to.)

Note
The name you choose for your Tenant will be used to identify data read from Dynamics CRM. You can change this value later. See section 8.1.1 for more information.

1.2.2 Add the connection string for the CRM

Dynamics CRM Connect uses a connection string to specify which CRM to connect to.

Talk to your CRM administrator to get the proper value.

Note
Documentation on the supported format for this connection string is available from the Microsoft Developer Network (MSDN): https://msdn.microsoft.com/en-us/library/gg695810(v=crm.7).aspx#parameters

Note
Dynamics CRM Connect does not currently support authentication using OAuth.

You must add the connection to the Sitecore connection strings file, which is located in `[web root]/App_Config/ConnectionStrings.config`.

The name of the connection string can be whatever you like, but it is recommended you use the tenant name. For example:

```
        <add name="mycrm" connectionString="url=https://####
/XRMServices/2011/Organization.svc;user id=####;password=####;organization=###;authentication
type=2" />
```

1.2.3 Create a new Entity Repository Set

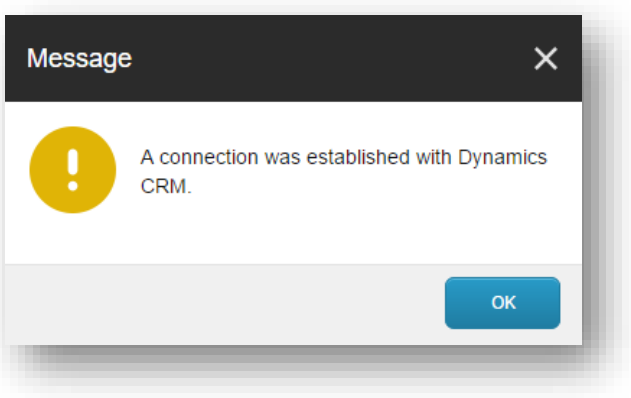
An Entity Repository Set is the component that controls the interactions with the CRM. You must create a new Entity Repository Set in order to read data from, and write data to, the CRM.

The Entity Repository Set is the object that uses the connection string you determined in section 1.2.2.

In Dynamics CRM Connect, an Entity Repository Set is configured using a Sitecore definition item. The section describes how to create this definition item.

1. Navigate to `/sitecore/system/Data Exchange/mycrm/Tenant Settings/Dynamics CRM/Repository Sets`.

2. Add the following item:
    a. **Template**: XRM Client Entity Repository Set
    b. **Name**: mycrm (You can use any value you want. This value should describe the CRM server you will connect to.)
3. In the field `Connection string name`, enter the connection string name for the Dynamics CRM server from section 1.2.2
4. Save the item.
5. Use the button `Test Connection` to test the connection. If a connection can be made, the message will indicate.



Note
If an error message is displayed, check the Sitecore log for more a more detailed description of the problem.

1.2.4 Configure the Endpoint

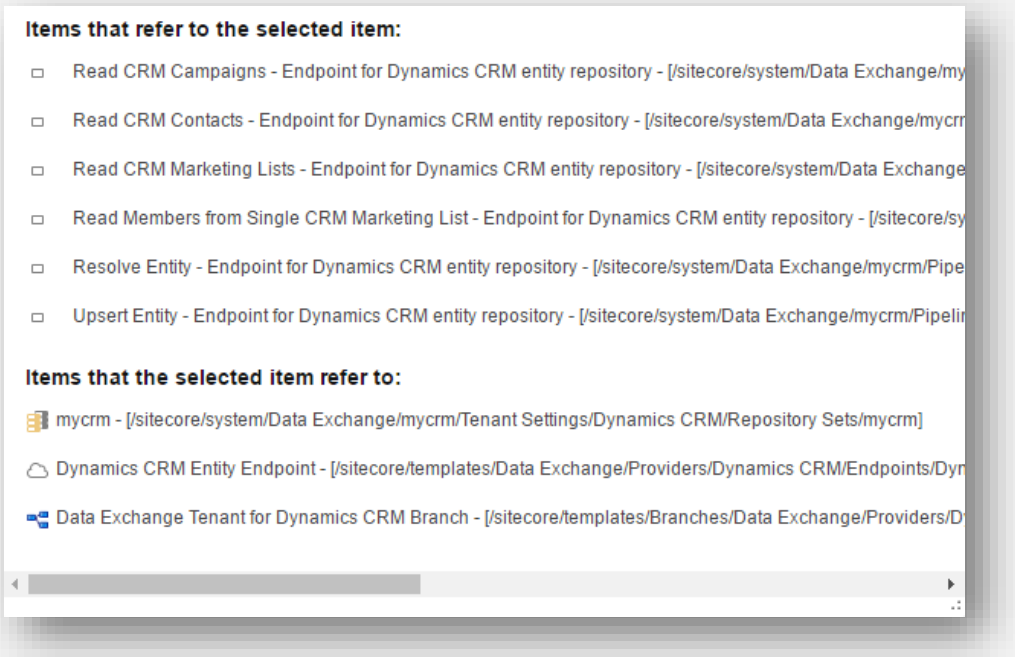
An Endpoint represents a system you can read data from or write data to. The Endpoint that represents your CRM system is already created. But it is not connected to an Entity Repository Set.

1. Navigate to `/sitecore/system/Data Exchange/mycrm/Endpoints/Providers/Dynamics CRM/Dynamics CRM Entity Endpoint`.
2. In the field `Entity repository set`, select the Entity Repository Set you created in section 1.2.3.
3. Save the item.

Note
It may seem redundant to have both an Entity Repository Set and an Endpoint. Why not have only an Endpoint?

Consider the configuration you just did. You had to create an Entity Repository Set and then assign the Entity Repository Set to an Endpoint. Imagine how this would be different if there was no Entity Repository Set, only an Endpoint.

The Endpoint is used in many places within the Tenant. If you look at the list of items that refer to the Endpoint you will see that there are 6 items that reference the Endpoint:



Without the Entity Repository Set, if you ever needed to change the Endpoint (for example, if you wanted to change from using a connection string to using a connection string name), you would need to create a new Endpoint and then change those 6 items.

Instead, all you need to do is Entity Repository Set on the endpoint. You are able to leave those 6 items untouched.

1.2.5 Enable the Tenant

By default, a Tenant is disabled when it is created. The Tenant must be enabled before it can be used to synchronize data.

1. Navigate to `/sitecore/system/Data Exchange/mycrm`.
2. In the field `Enabled`, tick the checkbox.
3. Save the item.

1.3 Synchronization

Synchronization involves making data from Dynamics CRM available in Sitecore, or making data from Sitecore available in Dynamics CRM. This section covers the former.

1.3.1 Starting a Pipeline Batch Manually

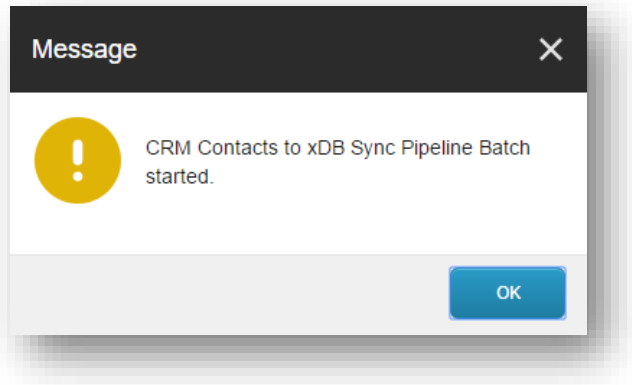
Data synchronization is controlled using items called Pipeline Batches. Pipeline Batches can be started in a number of ways. The easiest way is to start the Pipeline Batch manually.

Note
Be aware of the data in your CRM. If you run the synchronization process against a CRM with a large number of contacts, the synchronization process may take a long time to complete.

If you are only interested in testing to confirm the synchronization process is working, consider setting a limit on the number of contacts that will be handled. For more information on how to configure this, see section 3.2.4.

1. Navigate to the tenant "mycrm".
2. Navigate to `Pipeline Batches/CRM Contacts to xDB Sync Pipeline Batch`.
3. Click the button `Run Pipeline Batch`.

4. A message will appear to indicate the Pipeline Batch started.



1.3.2 Determining when the Pipeline Batch is finished running

The Pipeline Batch runs as an asynchronous task on the Sitecore server. This is done so that you can continue working in Sitecore as the Pipeline Batch runs.

But this also means that you will not be notified when the Pipeline Batch is finished running. There are, however, a couple of ways you can tell that the Pipeline Batch has completed:

Option 1

The `Run Pipeline Batch` button will be disabled while the Pipeline Batch is running. If the button is enabled, you know the Pipeline Batch is completed.

Note
This criterion is not 100% reliable. The Pipeline Batch already running is only one reason why the button may be disabled. Other reasons include the Tenant being disabled and no Pipelines being assigned to the Pipeline Batch.

Option 2

The Pipeline Batch item has `Summary` section. In this section there is a field `Last run finished`. If the value in this field has a date/time value that is equal to or later than the value in the field `Requested at`, you know the Pipeline Batch is completed.

Note
This criterion is not 100% reliable. If an unhandled exception is thrown while the Pipeline Batch is running, the Pipeline Batch will stop running. However, the `Last run finished` value will not be updated.

1.3.3 Confirming the data was synchronized

After the Pipeline Batch finishes, there are several ways you can confirm the data was synchronized.

Option 1

The Pipeline Batch item has `Summary` section. In this section there is a field `Last run completed`. If this field is ticked, the Pipeline Batch finished without any critical errors.

While this is not a fool-proof method - it is possible that a non-critical error occurred that prevented the data from being synchronized - it is a good first option to use.

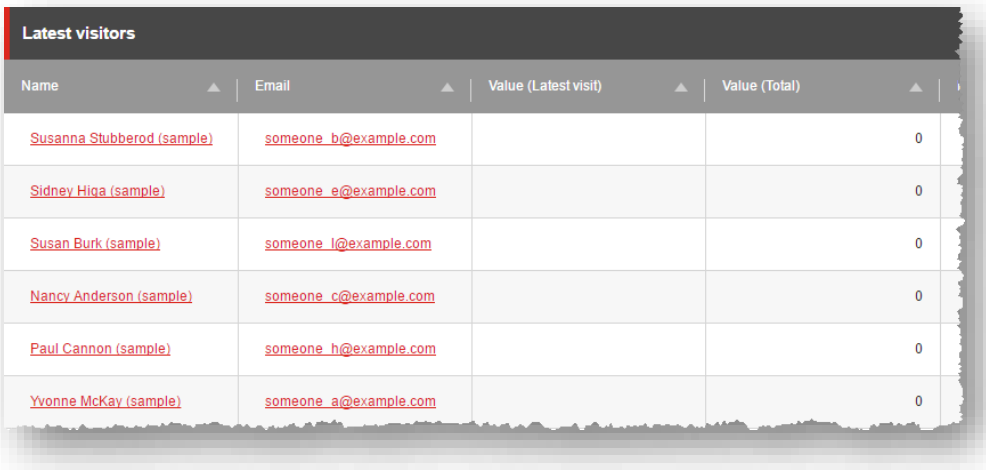
Options 2

Check the `Contacts` collection in MongoDB. You will find the active contacts from the CRM in the collection.

```
{
    "_id" : NUUID("556550ca-eaf0-4a9b-94b6-b7d56f1466fe"),
    "Identifiers" : {
        "Identifier" : "someone_m@example.com"
    },
    "Personal" : {
        "FirstName" : "Thomas",
        "Surname" : "Andersen (sample)"
    },
    "Emails" : {
        "Preferred" : "mycrm",
        "Entries" : {
            "mycrm" : {
                "SmtpAddress" : "someone_m@example.com"
            }
        }
    },
    "DynamicsCrm" : {
        "ContactId" : NUUID("40bc3e80-6408-e611-80da-5065f38ae991"),
        "LastSynced" : ISODate("2016-05-25T23:48:14.658Z"),
        "CrmName" : "mycrm"
    },
    "Lease" : null
}
```

Option 3

Open the Experience Profile. You will find the active contacts from the CRM in Experience Profile.



Latest visitors

Name	Email	Value (Latest visit)	Value (Total)
Susanna Stubberod (sample)	someone_b@example.com		0
Sidney Higa (sample)	someone_e@example.com		0
Susan Burk (sample)	someone_l@example.com		0
Nancy Anderson (sample)	someone_c@example.com		0
Paul Cannon (sample)	someone_h@example.com		0
Yvonne McKay (sample)	someone_a@example.com		0

1.3.4 Scheduling a Pipeline Batch

Dynamics CRM Connect includes commands that can be run using Sitecore's task scheduling features.

Tasks can be scheduled using Content Editor, under the item `/sitecore/system/Tasks/Schedules`.

When you install Dynamics CRM Connect, a folder is created at `/sitecore/system/Tasks/Commands/Data Exchange`. The following templates can be used to create commands that can be scheduled as tasks:

Command template	Description
Run All Pipeline Batches Command	Allows you to select a folder that contains Pipeline Batches. When the command is run, each of the Pipeline Batches in the folder are run in the order they appear under the folder.
Run Selected Pipeline Batches Command	Allows you to select specific Pipeline Batches. When the command is run, each of the selected Pipeline Batches is run in the order it was selected.

1.3.5 Starting a Pipeline Batch Programmatically

Pipeline Batches can be started programmatically. The following code demonstrates how to do this:

```
        Guid tenantId = ...       //identify the tenant that owns the pipeline batch
        Guid pipelineBatchId = ...  //identity the pipeline batch
        IEnumerable<PipelineBatch> batches = null;
        tasks =
Sitecore.DataExchange.Context.PipelineBatchRepository.GetPipelineBatches(tenantId);
        if (batches != null)
        {
         PipelineBatch pipelineBatch = batches.FirstOrDefault(x => x.ID == pipelineBatchId);
         if (pipelineBatch != null)
         {
           var runner = new InProcessPipelineBatchRunner();
           if (runner.Run(pipelineBatch))
           {
             //pipeline batch was started
           }
         }
        }
```

Note
Depending on the Pipeline Batch you run, it may take a long time for the task to complete. Unless you are certain that the Pipeline Batch will finish is a short amount of time, you may prefer to run the Pipeline Batch in a Sitecore task.

Chapter 2

CRM Campaign Synchronization

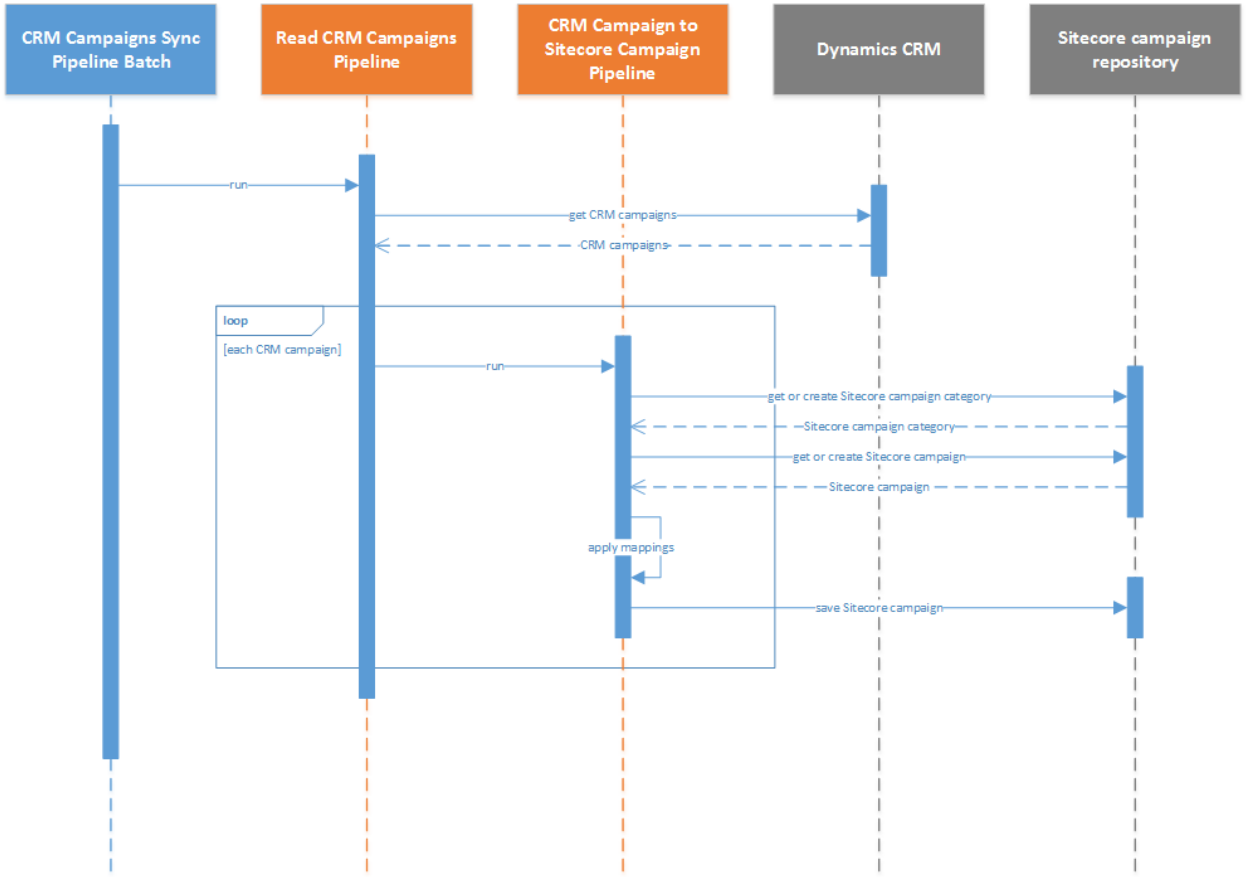
Dynamics CRM Connect comes with a pre-defined Pipeline Batch that reads campaigns from Dynamics CRM and creates campaigns in Sitecore that correspond to the campaigns that were read from CRM. This chapter describes how this Pipeline Batch works.

2.1 Overview

The campaign synchronization process involves the following steps:

1. Read campaigns from CRM.
2. Loop through the campaigns from CRM. For each campaign:
    a. Get the Sitecore campaign that corresponds to the CRM campaign if one exists, otherwise create a new Sitecore campaign
    b. Get the Sitecore campaign category that corresponds to the Tenant if one exists, otherwise create a new Sitecore campaign category
    c. Apply value mappings by reading values from the CRM campaign and writing those values to the Sitecore campaign
    d. Save Sitecore campaign

Note
Campaigns are not automatically deployed. You must manually deploy campaigns before they can be used in Sitecore.



Note
A detailed sequence diagram that fully documents the campaign synchronization process is available as a separate document along with the other product documentation.

2.1.1 Default value mappings

By default, the campaign synchronization process maps the following values from the CRM campaign to the Sitecore campaign. These mappings are defined in the Value Mapping Set named "CRM Campaign to Sitecore Campaign":

CRM campaign attribute	Sitecore campaign property	Description
campaignid	Id	The unique identifier for the campaign in the CRM.
name	Name	The campaign's name.
actualstart	StartDate	The campaign's start date.
actualend	EndDate	The campaign's end date.
createdon	CreatedDate	The date the campaign was created in the CRM.
modifiedon	LastModifiedDate	The date the campaign was last modified in the CRM.
description	Description	The campaign's description.

2.2 Common customizations

This section describes common customizations that you may need to make to the synchronization process.

2.2.1 Add fields to synchronize

The default value mappings are described in section 2.1.1. These mappings can be changed, and new mappings can be added.

In order to add a new field, you must know two values: one from Dynamics CRM and one from Sitecore.

Identity the value on the Dynamics CRM campaign you want to synchronize

The value from Dynamics CRM is the name of the attribute on the CRM campaign entity that is used to store the value. Your Dynamics CRM administrator is able to provide you with the names of the available attributes.

Identity the value on the Sitecore campaign you want to synchronize

The value from Sitecore is the name of the property on the campaign entity that is used to store the value. The available properties are the properties that can be set on the type `Sitecore.Marketing.Campaigns.Services.Model.CampaignEntity, Sitecore.Marketing.Campaigns.Services`.

Define Value Accessors

1. In Content Editor, navigate to your Tenant under `/sitecore/system/Data Exchange`.
2. Under the Tenant, navigate to `Data Access/Value Accessor Sets/Providers/Dynamics CRM/CRM Campaign Attributes`.
3. Insert a new item using the template `Entity Attribute Value Accessor`.
4. In the field `Attribute name`, enter the name of the attribute you received from your Dynamics CRM administrator.
5. Save the item.
6. Navigate back to your Tenant.
7. Under the Tenant, navigate to `Data Access/Value Accessor Sets/Providers/Sitecore/Sitecore Campaign Entity Fields`.
8. Insert a new item using the template `Property Value Accessor`.
9. In the field `Property name`, enter the name of the property on the Sitecore campaign you want to set.
10. Save the item.

Define Value Mapping

1. Navigate back to your Tenant.
2. Under the Tenant, navigate to `Value Mapping Sets/CRM Campaign to Sitecore Campaign`.
3. Insert a new item using the template `Value Mapping`.
4. In the field `Value accessor for source object`, select the item you created under `Data Access/Value Accessor Sets/Providers/Dynamics CRM/CRM Campaign Attributes`.
5. In the field `Value accessor for target object`, select the item you created under `Data Access/Value Accessor Sets/Providers/Sitecore/Sitecore Campaign Entity Fields`.
6. Save the item.

2.2.2 Read inactive campaigns from Dynamics CRM

By default, only active campaigns are read from Dynamics CRM. In order to include inactive campaigns, you must do the following:

1. In Content Editor, navigate to your Tenant under `/sitecore/system/Data Exchange`.
2. Navigate to `Pipelines/CRM Campaign Pipelines/Read CRM Campaigns Pipeline/Read CRM Campaigns`.
3. In the field `Exclude active filter`, tick the box.
4. Save the item.

2.2.3 Prevent campaigns for being bucketed

Dynamics CRM campaign are exposed as Sitecore campaigns that are created under a Campaign Category item whose name matches the Tenant used to define the synchronization process.

Since it is possible that a large number of CRM campaigns are synchronized, Dynamics CRM Connect makes this Campaign Category item bucketable. This results in the individual Campaign items being added to an item bucket.

If you do not want the Campaign items to be added to an item bucket, you must do the following:

1. In Content Editor, navigate to your Tenant under `/sitecore/system/Data Exchange`.
2. Navigate to `Pipelines/CRM Campaign Pipelines/Read CRM Campaigns Pipeline/ CRM Campaign to Sitecore Campaign Pipeline/Resolve Campaign Category`.
3. In the field `Make campaign category item bucketable`, untick the box.
4. Save the item.
5. Navigate to `/sitecore/system/Marketing Control Panel/Campaigns`.
6. If a Campaign Category item exists for your Tenant, delete that item.

Note
If prompted with a warning about breaking links, select the option to leave links. This is important because, when you re-run the campaign synchronization process, the campaign items are created with the same Sitecore item ID as the campaign items you deleted in the previous step. Leaving links ensures that anywhere those campaigns are used will continue to work as expected.

7. Navigate to your Tenant under `/sitecore/system/Data Exchange`.
8. Navigate to `Pipelines Tasks/CRM Campaigns Sync Pipeline Batch`.
9. Run the Pipeline Batch.

2.2.4 Filter Dynamics CRM campaigns using an expression

By default, all active campaigns are read from Dynamics CRM. You can define an expression that provides a greater amount of control over which campaigns are read.

Note
This setting is used in addition to the option to include inactive campaigns described in section 2.2.2.

1. In Content Editor, navigate to your Tenant under `/sitecore/system/Data Exchange`.
2. Navigate to `Tenant Settings/Dynamics CRM/Filter Expressions/Campaign Filter Expressions`.
3. Insert a new item using the template `Filter Expression`.

4. Configure the item. For details on how to configure expressions, see section 9.1.
5. Navigate back to your Tenant.
6. Under the Tenant, navigate to `Pipelines/CRM Campaign Pipelines/Read CRM Campaigns Pipeline/Read CRM Campaigns`.
7. In the field `Filter expression`, select the expression you created.
8. Save the item.

2.2.5 Limit to a certain number of Dynamics CRM campaigns

By default, all active campaigns are read from Dynamics CRM. As described above, it is possible to include inactive campaigns, as well as filter campaigns using an expression. You can also set a limit for the number of campaigns that are handled.

Note
This setting is used in addition to the option to include inactive campaigns described in section 2.2.2 and the option to use an expression described in section 2.2.3.

Note
This setting does not restrict the number of campaigns that are read from Dynamics CRM. The number of campaigns that are read is determined by a combination of the expression that is used and the page size that is specified.

1. In Content Editor, navigate to your Tenant under `/sitecore/system/Data Exchange`.
2. Navigate to `Pipelines/CRM Campaign Pipelines/Read CRM Campaigns Pipeline/Read CRM Campaigns`.
3. In the field `Maximum number of entities to read`, enter the maximum number of campaigns you want to handle.
4. Save the item.

Chapter 3

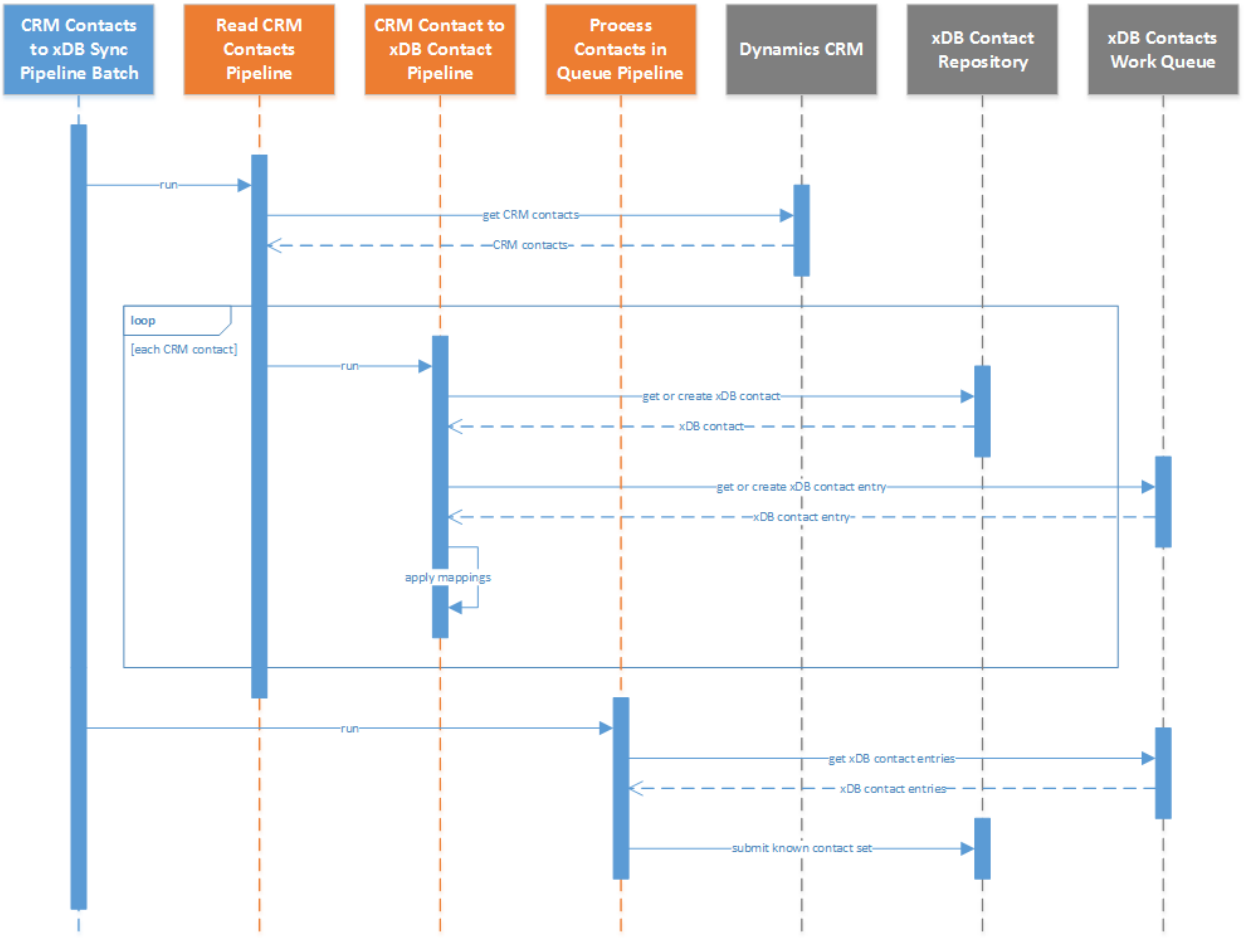
CRM Contact Synchronization

Dynamics CRM Connect comes with a pre-defined Pipeline Batch that reads contacts from Dynamics CRM and creates contacts in Sitecore that correspond to the contacts that were read from CRM. This chapter describes how this Pipeline Batch works.

3.1 Overview

The contact synchronization process involves the following steps:

1. Read contacts from CRM.
2. Loop through the contacts from CRM. For each contact:
    a. Get the xDB contact that corresponds to the CRM contact if one exists, otherwise create a new xDB contact
    b. Get the contact entry from the work queue that corresponds to the xDB contact if one exists, otherwise create a new work queue entry
    c. Apply value mappings by reading values from the CRM contact and writing those values to the work queue contact
3. Read contacts from work queue.
4. Create a known contact set for the xDB bulk contact update API.
5. Loop through the contacts from the work queue. For each contact:
    a. Add the contact to the known contact set
6. Submit the known contact set.



Note
A detailed sequence diagram that fully documents the contact synchronization process is available as a separate document along with the other product documentation.

3.1.1 Default value mappings

By default, the contact synchronization process maps the following values from the Dynamics CRM contact to the Sitecore contact. These mappings are defined in the Value Mapping Set named "CRM Contact to xDB Contact":

CRM contact attribute	xDB contact facet	Property on contact facet	Description
contactid	DynamicsCrm	ContactId	The unique identifier for the contact in the CRM.
address1_city	Addresses	City	The contact's city.
address1_country	Addresses	Country	The contact's country.
address1_line1	Addresses	StreetLine1	First line of the contact's address.
address1_line2	Addresses	StreetLine2	Second line of the contact's address.
address1_line3	Addresses	StreetLine3	Third line of the contact's address.
address1_postalcode	Addresses	PostalCode	The contact's postal code.
address1_stateorprovince	Addresses	StateProvince	The contact's state or province.
donotbulkemail	Communication Profile	ConsentRevoked	A value that indicates whether or not the contact has opted-out of bulk email.
emailaddress1	Emails	SmtpAddress	The contact's email address.
firstname	Personal	FirstName	The contact's first name.
jobtitle	Personal	JobTitle	The contact's job title.
lastname	Personal	Surname	The contact's last name (surname).
telephone1	Phone Numbers	Number	The contact's telephone number.

3.2 Common customizations

This section describes common customizations that you may need to make to the synchronization process.

3.2.1 Add fields to synchronize

The default value mappings are described in section 3.1.1. These mappings can be changed, and new mappings can be added.

In order to add a new field, you must know two values: one from Dynamics CRM and one from Sitecore.

Identity the value on the Dynamics CRM contact you want to synchronize

The value from Dynamics CRM is the name of the attribute on the CRM contact entity that is used to store the value. Your Dynamics CRM administrator is able to provide you with the names of the available attributes.

Identity the value on the xDB contact you want to synchronize

Values are stored on xDB contacts through contact facets. The contact facets that are available on your Sitecore server depend on how your system has been configured. Your Sitecore administrator is able to provide you with the information about the available contact facets and the values stored on each.

Define Value Accessor for Dynamics CRM attribute

1. In Content Editor, navigate to your Tenant under `/sitecore/system/Data Exchange`.
2. Under the Tenant, navigate to `Data Access/Value Accessor Sets/Providers/Dynamics CRM/CRM Contact Attributes`.
3. Insert a new item using the template `Entity Attribute Value Accessor`.
4. In the field `Attribute name`, enter the name of the attribute you received from your Dynamics CRM administrator.
5. Save the item.

Define Value Accessor for xDB Contact Facet

1. In Content Editor, navigate to your Tenant under `/sitecore/system/Data Exchange`.
2. Under the Tenant, navigate to `Data Access/Value Accessor Sets/Providers/Sitecore/xDB Contact Properties`.
3. Insert a new item using the template that is appropriate for the xDB Contact Facet value.

Note
Detailed descriptions of the available templates are available in section 9.2.

4. Configure the Value Accessor item.
5. Save the item.

Define Value Mapping

1. Navigate back to your Tenant.
2. Under the Tenant, navigate to `Value Mapping Sets/CRM Contact to xDB Contact`.
3. Insert a new item using the template `Value Mapping`.
4. In the field `Value accessor for source object`, select the item you created under `Data Access/Value Accessor Sets/Providers/Dynamics CRM/CRM Contact Attributes`.
5. In the field `Value accessor for target object`, select the item you created under `Data Access/Value Accessor Sets/Providers/Sitecore/xDB Contact Properties`.

6. Save the item.

3.2.2 Disable existing mappings

As described in section 3.1.1, many value mappings are configured by default for the CRM contact synchronization process. The more values that are mapped, the more data must be read from Dynamics CRM and written to xDB.

Existing value mappings can be disabled or deleted. However, disabling a value mapping is often preferable to deleting a value mapping because you may, in the future, decide that you want the value mapping after all.

Note
Disabling an existing mapping will not result in the data being removed from xDB. This will only prevent the data from being updated on existing contacts, and from being added to newly synchronized contacts.

Disable a Value Mapping

1. In Content Editor, navigate to your Tenant under `/sitecore/system/Data Exchange`.
2. Under the Tenant, navigate to `Value Mapping Sets/CRM Contact to xDB Contact`.
3. Select the Value Mapping item you want to disable.
4. Untick the value for the field `Enabled`.
5. Save the item.

Prevent the value from being read from Dynamics CRM

By disabling the Value Mapping, you will prevent the value from being set on the xDB contact. You will not, however, prevent Dynamics CRM Connect from reading the value from the CRM.

In order to ensure the value does not get read from the CRM, you must disable the Value Accessor that corresponds to the value.

1. In Content Editor, navigate to your Tenant under `/sitecore/system/Data Exchange`.
2. Under the Tenant, navigate to `Data Access/Value Accessor Sets/Providers/Dynamics CRM/CRM Contact Attributes`.
3. Select the Value Accessor.
4. Untick the value for the field `Enabled`.
5. Save the item.

3.2.3 Read inactive contacts from Dynamics CRM

By default, only active contacts are read from Dynamics CRM. In order to include inactive contacts, you must do the following:

1. In Content Editor, navigate to your Tenant under `/sitecore/system/Data Exchange`.
2. Navigate to `Pipelines/CRM Contact Pipelines/Read CRM Contacts Pipeline/Read CRM Contacts`.
3. In the field `Exclude active filter`, tick the box.
4. Save the item.

3.2.4 Limit to a certain number of Dynamics CRM contacts

By default, all active contacts are read from Dynamics CRM. As described above, it is possible to include inactive contacts, as well as filter contacts using an expression. You can also set a limit for the number of contacts that are handled.

Note
This setting is used in addition to the option to include inactive contacts described in section 3.2.3 and the option to use an expression described in section 3.2.3.

Note
This setting does not restrict the number of contacts that are read from Dynamics CRM. The number of contacts that are read is determined by a combination of the expression that is used and the page size that is specified.

1. In Content Editor, navigate to your Tenant under `/sitecore/system/Data Exchange`.
2. Navigate to `Pipelines/CRM Campaign Pipelines/Read CRM Contacts Pipeline/Read CRM Contacts`.
3. In the field `Maximum number of entities to read`, enter the maximum number of campaigns you want to handle.
4. Save the item.

Chapter 4

CRM Marketing List Synchronization

Dynamics CRM Connect comes with a pre-defined Pipeline Batch that reads marketing list membership from Dynamics CRM and makes this information available on Sitecore contacts. This chapter describes how this Pipeline Batch works.

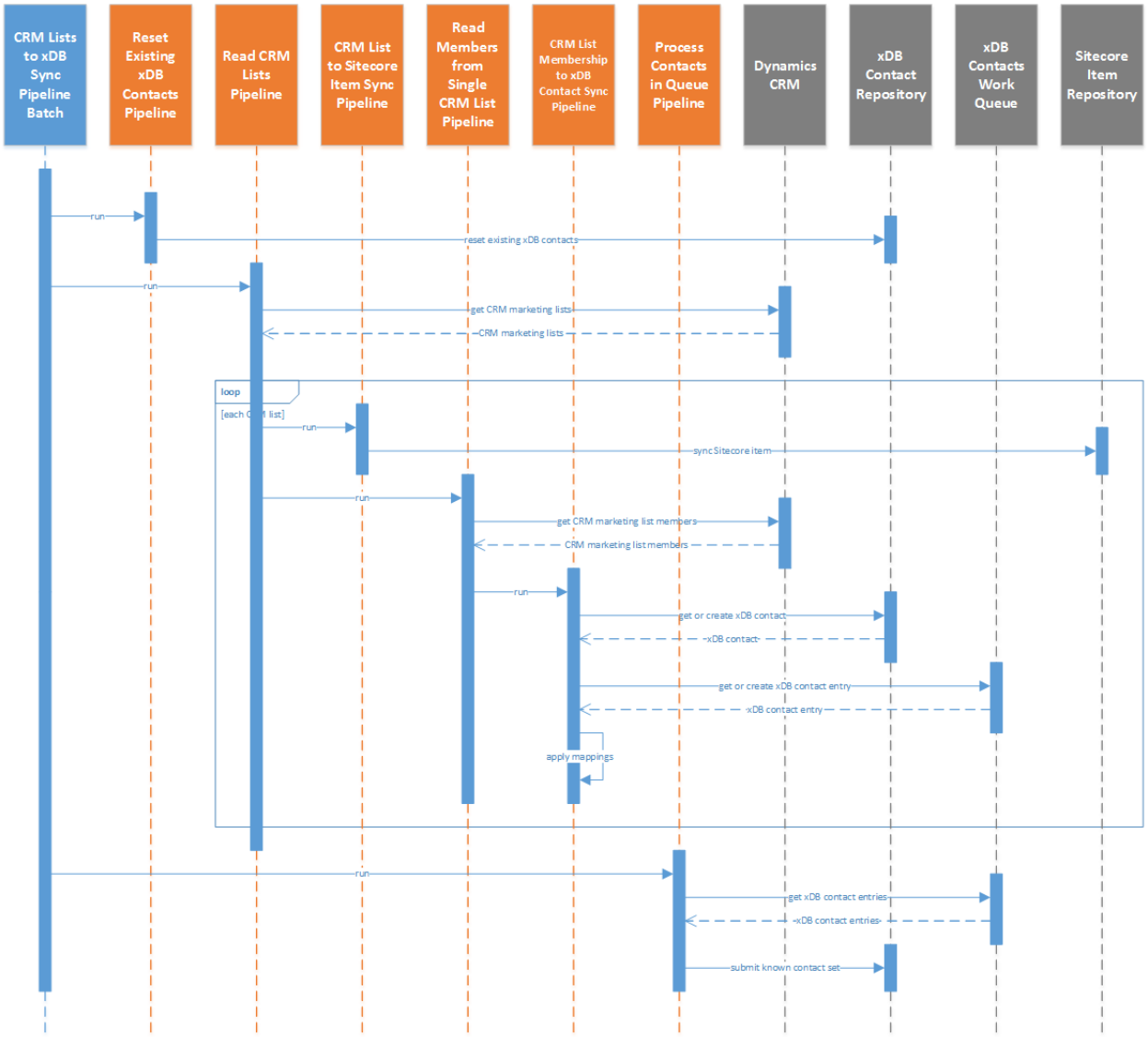
4.1 Overview

The marketing list synchronization process involves the following steps:

1. Reset marketing lists assigned to existing xDB contacts.

Note
For more information on why marketing lists are reset, see section 4.2.

2. Read marketing lists from CRM.
3. Loop through the marketing lists from CRM. For each marketing list:
    a. Get the Sitecore item that corresponds to the CRM marketing list if one exists, otherwise create a new Sitecore item
    b. Apply value mappings by reading values from the CRM marketing list and writing those values to the Sitecore item
    c. Save the Sitecore item
    d. Read the marketing list members from CRM
    e. Loop through the marketing list members from CRM. For each marketing list member:
        i. Get the xDB contact that corresponds to the CRM marketing list member if one exists, otherwise create a new xDB contact
        ii. Get the contact entry from the work queue that corresponds to the xDB contact if one exists, otherwise create a new work queue entry
        iii. Apply value mappings by reading values from the CRM marketing list member and writing those values to the work queue contact
4. Read contacts from work queue.
5. Create a known contact set for the xDB bulk contact update API.
6. Loop through the contacts from the work queue. For each contact:
    a. Add the contact to the known contact set
7. Submit the known contact set.



4.1.1 Default value mappings

By default, the marketing list synchronization process maps the following values from the Dynamics CRM marketing list to the Sitecore item that represents the marketing list. These mappings are defined in the Value Mapping Set named "CRM List to Sitecore Item":

CRM list attribute	Sitecore template	Sitecore field	Description
Listid	/sitecore/templates/Data Exchange/ Providers/Sitecore/Entities/ External Marketing List	MarketingListId	The unique identifier for the marketing list in the CRM.
Listname	/sitecore/templates/Data Exchange/ Providers/Sitecore/Entities/ External Marketing List	MarketingListName	The marketing list's name.

The marketing list synchronization process also maps the following values from the Dynamics CRM contacts that are members of the CRM marketing lists. These mappings are defined in the Value Mapping Set named "CRM List Membership to xDB Contact":

CRM list attribute	xDB contact facet	Property on contact facet	Description
listid	DynamicsCrm	MarketingLists	A comma-separated list of the unique identifiers for the CRM marketing lists the contact is a member of.

4.2 About Reset Marketing Lists

The step in the CRM marketing list synchronization process where existing marketing lists are reset is needed due to the way marketing list membership is read.

A simplified description of the logic is:

1. Read all marketing lists
2. For each marketing list, get the contacts who are members of the list
3. For each contact, add the marketing list to the collection of marketing lists on the contact

An example is helpful to understand why the reset is necessary. Consider when a contact is a member of multiple marketing lists:

Step	Contact value (before)	Contact value (after)
Update with List A	-	List A
Update with List B	List A	List A, List B
Update with List C	List A, List B	List A, List B, List C

What happens if the contact is removed from List C? The next time the synchronization process runs, CRM reports the contact is a member of List A and List B. How does List C get removed from the xDB contact?

The reset marketing lists step ensures that the contact value is empty before the CRM data is applied:

Step	Contact value (before)	Contact value (after)
Update with List A	-	List A
Update with List B	List A	List A, List B

What happens if the reset marketing lists step does not run? The values from the previous synchronization remain.

Step	Contact value (before)	Contact value (after)
Update with List A	List A, List B, List C	List A, List B, List C
Update with List B	List A, List B, List C	List A, List B, List C

Why is this approach used to read marketing list membership?

In a word: performance. We found that the alternatives (such as reading the marketing list membership for a contact at the same time you read the other contact data) are much slower when dealing with more than a couple dozen contacts.

It is fully within your control to change how any synchronization process works. If you prefer to use a different approach, you can implement it.

Limitations of this approach

This step is not selective in which contacts are reset. If you are synchronizing with multiple Dynamics CRM systems, or if you are using filters to limit the contacts that you are synchronizing, this part of the synchronization process may not work as expected.

Note
Future versions of Dynamics CRM Connect will allow you to assign a filter to the step that reads contacts from xDB so that only contacts that have been synchronized with a specific Dynamics CRM server will be selected.

4.3 Common customizations

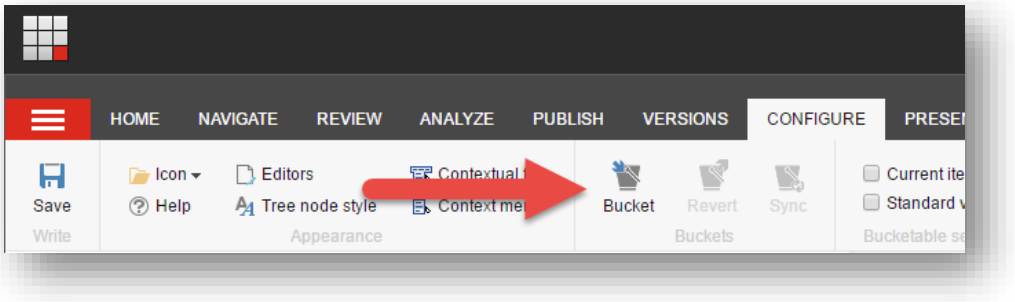
This section describes common customizations that you may need to make to the synchronization process.

4.3.1 Handling Large Numbers of Marketing Lists

By default, all CRM marketing lists for a tenant are stored as children under a single item.

If you have more than 100 marketing lists in CRM that you are bringing into Sitecore, it is recommended you set this item to be "bucketable". This will ensure Sitecore's item bucket feature is used, which will improve performance and user experience within Sitecore.

1. In Content Editor, navigate to your Tenant under `/sitecore/system/Data Exchange`.
2. Navigate to `Tenant Settings/Dynamics CRM/Marketing Lists`.
3. In the Ribbon, under the tab "Configure", click the button "Bucket".



4. Any child items already created will be moved into the item bucket structure. Any new items created (as a result of future synchronization) will be created in the item bucket structure.

Chapter 5

CRM Full Contact Synchronization

Dynamics CRM Connect comes with a pre-defined Pipeline Batch that reads both contacts and marketing list membership from Dynamics CRM, creates contacts in Sitecore that correspond to the contacts that were read from CRM, and associates the Sitecore contact with the CRM marketing lists that were read from CRM. This chapter describes how this Pipeline Batch works.

5.1 Overview

The CRM Full Contact Synchronization process runs the CRM Contact Synchronization process and the CRM Marketing List Synchronization process as a single synchronization process.

Both the CRM Contact Synchronization and the CRM Marketing List Synchronization processes update xDB Contacts. Both of these processes update contacts work by storing the updated contacts in a work queue, and only updating xDB after all of the contacts are read. This reduces the number of times you have to explicitly write to xDB, which may improve performance.

The CRM Full Contact Synchronization process uses this same approach. It only updates xDB after the contacts are synchronized, and the marketing lists are synchronized. Again, the benefit is that, in general, the fewer times you have to write to xDB, the better the performance on your Sitecore servers.

5.2 Common customizations

The customizations described in section 3.2 and section 4.3 also apply to this synchronization process.

Note
This Pipeline Batch uses the same Pipelines as the CRM Contacts to xDB Sync Pipeline Batch and the CRM Lists to xDB Sync Pipeline Batch. This means any changes you make to this Pipeline Batch may affect these other Pipeline Batches.

Chapter 6

Sitecore Contact Synchronization

Dynamics CRM Connect comes with a pre-defined Pipeline Batch that reads contacts from Sitecore Experience Database and updates contacts in Dynamics CRM that correspond to the contacts that were read from Sitecore. This chapter describes how this Pipeline Batch works.

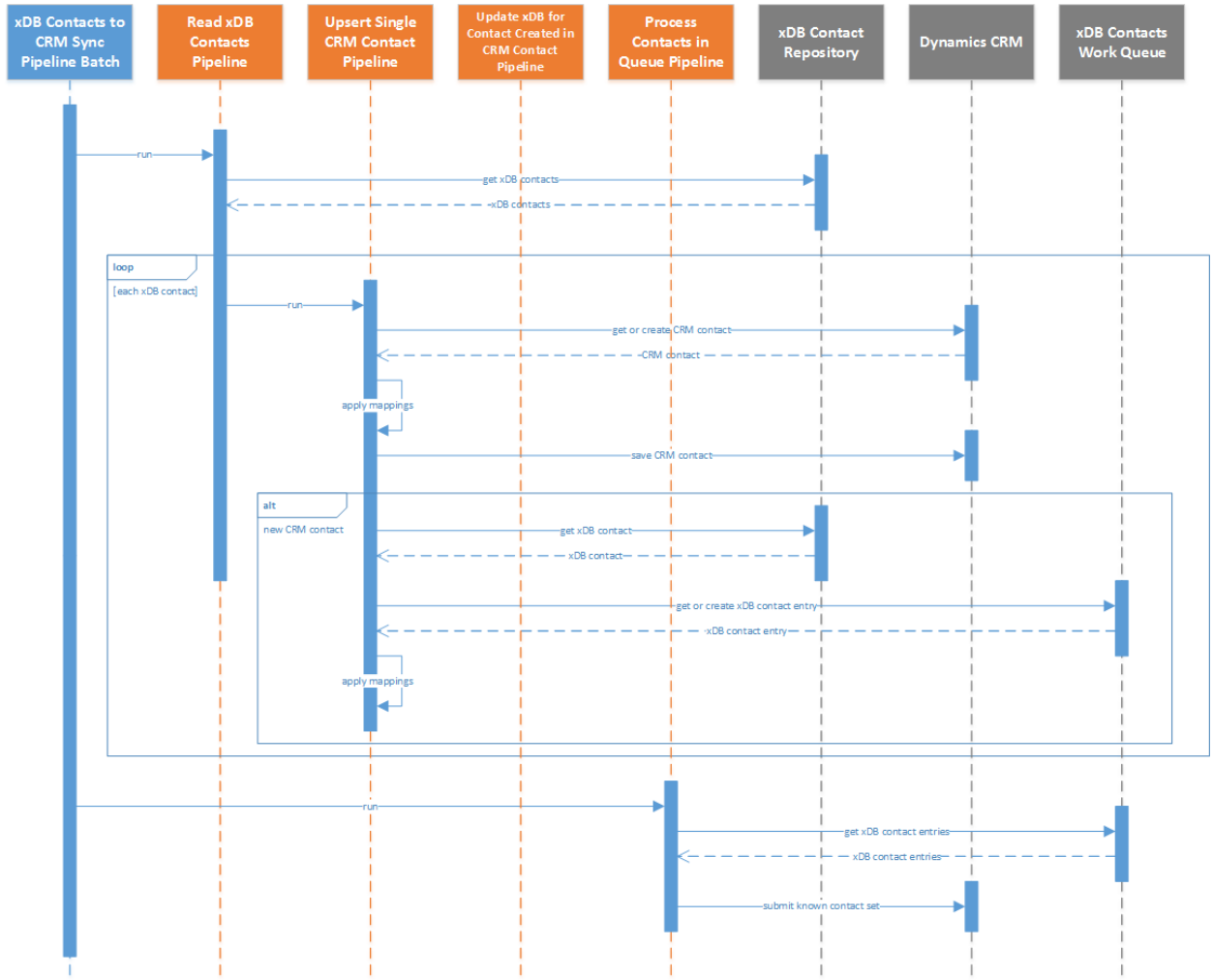
6.1 Overview

The contact synchronization process involves the following steps:

1. Read contacts from xDB.
2. Loop through the contacts from xDB. For each contact:
    a. Get the CRM contact that corresponds to the xDB contact if one exists, otherwise create a new CRM contact
    b. Apply value mappings by reading values from the xDB contact and writing those values to the CRM contact
    c. Save the CRM contact
    d. If the CRM contact was created
        i. Get the xDB contact
        ii. Get the contact from the work queue that corresponds to the xDB contact if one exists, otherwise add the contact to the work queue
        iii. Apply value mappings by reading values the CRM contact and writing those values to the work queue contact

Note
Writing data back to the xDB contact is needed so that the CRM contact id is set on the xDB contact. This facilitates future synchronization.

3. Read contacts from work queue.
4. Create a known contact set for the xDB bulk contact update API.
5. Loop through the contacts from the work queue. For each contact:
    a. Add the contact to the known contact set
6. Submit the known contact set.



6.1.1 Default value mappings

By default, the contact synchronization process maps the following values from the Sitecore contact to the Dynamics CRM contact:

xDB contact facet	Property on contact facet	CRM contact attribute	Description
Emails	SmtpAddress	emailaddress1	The contact's email address.
Personal	FirstName	firstname	The contact's first name.
Personal	Surname	lastname	The contact's last name (surname).

For contacts that are created in Dynamics CRM, the following values are mapped back to the Sitecore contact:

CRM contact attribute	xDB contact facet	Property on contact facet	Description
contactid	DynamicsCrm	ContactId	The unique identifier for the contact in the CRM.
[not applicable]	DynamicsCrm	CrmName	The name of the Sitecore tenant.

6.2 Common customizations

This section describes common customizations that you may need to make to the synchronization process.

6.2.1 Add fields to synchronize

Adding fields to synchronize is accomplished using the same process described in section 3.2.1

6.2.2 Disable existing mappings

Disabling existing mappings is accomplished using the same process described in section 3.2.2.

Chapter 7

Supporting Additional CRM Entities

Dynamics CRM Connect is built on a highly adaptable and extensible framework. It is possible to support virtually any entity from Dynamics CRM.

In many cases this can be accomplished without any custom development. This is referred to as "simple synchronization" in this document. Other cases require custom development. This is referred to as "complex synchronization". Both types of synchronization are covered in this chapter.

7.1 Simple Synchronization

No custom development is needed to read entities from CRM and to represent those entities in Sitecore, provided Sitecore already has a data structure that can be used to store the data. In this example, we are reading entities from CRM (accounts) and are representing those entities in an existing data structure (Sitecore items).

Associating an xDB contact with a CRM account will require custom code, because it is doing more than reading entities (it is reading a relationship between the contact and account entities) and is storing that data in a data structure that must be created (a custom contact facet). Section 7.2 describes how to implement this.

7.1.1 Confirm CRM Privileges

In order for Dynamics CRM Connect to be able to access account information from CRM, the CRM user used to connect to CRM must have the following permissions assigned:

- Read account entities

If these permissions are not assigned, an error will occur when Sitecore tries to read data from CRM.

Your CRM administrator can confirm that the CRM user has the required permissions.

7.1.2 Add Templates for CRM Entity Data

You have configured the ability for an xDB contact to be associated with a CRM account. Now you must create a Sitecore template that can be used to store information about the CRM account.

Having Sitecore items that represent CRM accounts will allow your users to configure rules based on account membership. For example, a marketer will be able to personalize a web page based on whether or not a visitor is associated with a specific account.

The marketer must be able to select from a list of available accounts. This is the purpose of exposing CRM accounts as Sitecore items.

1. In Sitecore, open Template Manager.
2. Navigate to `Templates/Data Exchange/Providers/Dynamics CRM/Folders`.
3. Add a new template named "Accounts Folder".
4. Set the icon for this template to `Office/32x32/folder_open.png`.
5. Navigate to `Templates/Data Exchange/Providers/Sitecore/Entities`.
6. Add a new template named "External Account".
7. Set the icon for this template to `Office/32x32/office_building2.png`.
8. Add a section named "Data".
9. Add the following field:
    a. **Name**: AccountName
    b. **Type**: Single-Line Text
    c. **Shared**: ticked
10. Navigate to `Templates/Data Exchange/Providers/Sitecore/Entities/External Account/Data/AccountName`.
11. Set the following field value:
    a. **Field**: Title
    b. **Value**: Account name

12. Save the item.
13. Add the following field:
    a. **Name**: AccountId
    b. **Type**: Single-Line Text
    c. **Shared**: ticked
14. Navigate to `Templates/Data Exchange/Providers/Sitecore/Entities/External Account/Data/AccountId`.
15. Set the following field value:
    a. **Field**: Title
    b. **Value**: Account ID
16. Save the item.
17. Open Content Editor.
18. Navigate to the tenant "mycrm".
19. Navigate to `Tenant Settings/Dynamics CRM`.
20. Add the following item:
    a. **Template**: /Data Exchange/Providers/Dynamics CRM/Folders/Accounts Folder
    b. **Name**: Accounts

7.1.3 Add Entity Repository

Each type of CRM entity you want to interact with must be defined on the Entity Repository Set that your Dynamics CRM Endpoint uses.

Note
These instructions assume you have an entity repository set named "mycrm entity repository set". The name of your repository set may be different, so you may need to adjust some of the paths accordingly.

1. In Sitecore, open Content Editor.
2. Navigate to the tenant "mycrm".
3. Navigate to `Tenant Settings/Dynamics CRM/Repository Sets/mycrm entity repository set`.
4. Add the following item:
    a. **Template**: XRM Client Entity Repository
    b. **Name**: account

7.1.4 Add Value Accessor Set for CRM Entity

In order to be able to read attributes from a CRM account entity, a Value Accessor Set is needed. This item identifies the attributes from the CRM account that are available to Dynamics CRM Connect.

1. Navigate to the tenant "mycrm".
2. Navigate to `Data Access/Value Accessor Sets/Providers/Dynamics CRM`.
3. Add the following item:
    a. **Template**: Entity Value Accessor Set
    b. **Name**: CRM Account Attributes

4. Navigate to `CRM Account Attributes`.
5. Add the following item:
    a. **Template**: Entity Attribute Value Accessor
    b. **Name**: Account Id
6. Set the following field value:
    a. **Field name**: Attribute name
    b. **Value**: accountid
7. Save the item.
8. Navigate to `CRM Account Attributes`.
9. Add the following item:
    a. **Template**: Entity Attribute Value Accessor
    b. **Name**: Account Name
10. Set the following field value:
    a. **Field name**: Attribute name
    b. **Value**: name
11. Save the item.

7.1.5 Add Value Accessor Set for Sitecore Item

In order to be able to write values to the field on a Sitecore account item, a Value Accessor Set is needed. This item identifies the fields from the Sitecore account that are available to Dynamics CRM Connect.

1. Navigate to the tenant "mycrm".
2. Navigate to `Data Access/Value Accessor Sets/Providers/Sitecore`.
3. Add the following item:
    a. **Template**: Sitecore Item Field Value Accessor Set
    b. **Name**: Sitecore Account Item Fields
4. Navigate to `Sitecore Account Item Fields`.
5. Add the following item:
    a. **Template**: Sitecore Item Field Value Accessor
    b. **Name**: Account Id
6. Set the following field value:
    a. **Field name**: Field
    b. **Value**: Templates/Data Exchange/Providers/Sitecore/Entities/External Account/Data/AccountId
7. Save the item.
8. Navigate to `Sitecore Account Item Fields`.
9. Add the following item:
    a. **Template**: Sitecore Item Field Value Accessor
    b. **Name**: Account Name

10. Set the following field value:
    a. **Field name**: Field
    b. **Value**: Templates/Data Exchange/Providers/Sitecore/Entities/External Account/Data/AccountName
11. Save the item.

7.1.6 Add Value Mapping Set

In order to map values from the source object (CRM account entity) to a target object (Sitecore account item), you must define a Value Mapping Set.

1. Navigate to the tenant "mycrm".
2. Navigate to `Value Mapping Sets`.
3. Add the following item:
    a. **Template**: Value Mapping Set
    b. **Name**: CRM Account to Sitecore Item
4. Navigate to `CRM Account to Sitecore Item`.
5. Add the following item:
    a. **Template**: Value Mapping
    b. **Name**: Account Id
6. Set the following field value:
    a. **Field name**: Value accessor for source object
    b. **Value**: Data Access/Value Accessor Sets/Providers/Dynamics CRM/CRM Account Attributes/Account Id
7. Set the following field value:
    a. **Field name**: Value accessor for target object
    b. **Value**: Data Access/Value Accessor Sets/Providers/Sitecore/Sitecore Account Item Fields/Account Id
8. Save the item.
9. Navigate to `CRM Account to Sitecore Item`.
10. Add the following item:
    a. **Template**: Value Mapping
    b. **Name**: Account Name
11. Set the following field value:
    a. **Field name**: Value accessor for source object
    b. **Value**: Data Access/Value Accessor Sets/Providers/Dynamics CRM/CRM Account Attributes/Account Name
12. Set the following field value:
    a. **Field name**: Value accessor for target object
    b. **Value**: Data Access/Value Accessor Sets/Providers/Sitecore/Sitecore Account Item Fields/Account Name
13. Save the item.

7.1.7 Add Pipelines to Handle CRM Entity

Each account entity read from CRM is represented in Sitecore by an item. A Pipeline is used to define the logic involved in creating the Sitecore item.

Note
You have not yet configured the Pipeline that will read the entities from CRM. The Pipeline that handles each entity is used in the Pipeline that reads the entities, so the Pipeline that handles each entity is configured first.

1. Navigate to the tenant "mycrm".
2. Navigate to `Pipelines`.
3. Add the following item:
    a. **Template**: Pipelines Folder
    b. **Name**: CRM Account Pipelines
4. Navigate to `CRM Account Pipelines`.
5. Add the following item:
    a. **Template**: Pipeline
    b. **Name**: CRM Account to Sitecore Account Sync Pipeline
6. Navigate to `CRM Account to Sitecore Account Sync Pipeline`.
7. Add the following item:
    a. **Template**: Resolve Sitecore Item Pipeline Step
    b. **Name**: Resolve Sitecore Item
8. Set the following field value:
    a. **Field name**: Template for new item
    b. **Value**: Templates/Data Exchange/Providers/Sitecore/Entities/External Account
9. Set the following field value:
    a. **Field name**: Value accessor for name for new item
    b. **Value**: Data Access/Value Accessor Sets/Providers/Dynamics CRM/CRM Account Attributes/Account Name
10. Set the following field value:
    a. **Field name**: Endpoint to read data from
    b. **Value**: Sitecore/Sitecore Item Model Repository Endpoint
11. Set the following field value:
    a. **Field name**: Identifier value accessor
    b. **Value**: Value Accessor Sets/Providers/Dynamics CRM/CRM Account Attributes/Account Id
12. Set the following field value:
    a. **Field name**: Identifier object location
    b. **Value**: Pipeline Processor Context Source
13. Set the following field value:
    a. **Field name**: Resolved object location
    b. **Value**: Pipeline Processor Context Target

14. Set the following field value:
    a. **Field name**: Parent for item to resolve
    b. **Value**: sitecore/system/Data Exchange/mycrm/Tenant Settings/Dynamics CRM/Accounts
15. Set the following field value:
    a. **Field name**: Value accessor for Sitecore item field used to match the identifier value
    b. **Value**: Data Access/Value Accessor Sets/Providers/Sitecore/Sitecore Account Item Fields/Account Id
16. Save the item.
17. Navigate to `/sitecore/system/Data Exchange/mycrm/Pipelines/CRM Account Pipelines/CRM Account to Sitecore Account Sync Pipeline`.
18. Add the following item:
    a. **Template**: Apply Mapping Pipeline Step
    b. **Name**: Apply Mapping
19. Set the following field value:
    a. **Field name**: Mapping set
    b. **Value**: Value Mapping Sets/CRM Account to Sitecore Item
20. Save the item.
21. Navigate to `CRM Account to Sitecore Account Sync Pipeline`.
22. Add the following item:
    a. **Template**: Update Sitecore Item Pipeline Step
    b. **Name**: Update Sitecore Item
23. Set the following field value:
    a. **Field name**: Endpoint for Sitecore item model repository
    b. **Value**: Sitecore/Sitecore Item Model Repository Endpoint
24. Save the item.
25. Navigate to `CRM Account to Sitecore Account Sync Pipeline`.
26. Change the order of the child items to the following:
    a. Resolve Sitecore Item
    b. Apply Mapping
    c. Update Sitecore Item

7.1.8 Add Pipelines to Read CRM Entities

A Pipeline is needed to read accounts from CRM and to pass each account to the pipeline configured in section 7.1.7.

1. Navigate to the tenant "mycrm".
2. Navigate to `Pipelines/CRM Account Pipelines`.
3. Add the following item:
    a. **Template**: Pipeline
    b. **Name**: Read CRM Accounts Pipeline

4. Navigate to `Read CRM Accounts Pipeline`.
5. Add the following item:
    a. **Template**: Read CRM Entities Pipeline Step
    b. **Name**: Read CRM Accounts
6. Set the following field value:
    a. **Field name**: Endpoint for Dynamics CRM entity repository
    b. **Value**: Dynamics CRM/Dynamics CRM Entity Endpoint
7. Set the following field value:
    a. **Field name**: Entity name
    b. **Value**: account
8. Set the following field value:
    a. **Field name**: Attributes to read
    b. **Value**: Common CRM Entity Attributes, CRM Account Attributes
9. Save the item.
10. Navigate to `Read CRM Accounts Pipeline`.
11. Add the following item:
    a. **Template**: Iterate Data and Run Pipelines
    b. **Name**: Iterate CRM Accounts and Run Pipelines
12. Set the following field value:
    a. **Field name**: Pipelines
    b. **Value**: CRM Account to Sitecore Account Sync Pipeline
13. Save the item.
14. Navigate to `Read CRM Accounts Pipeline`.
15. Reorder the child items so they are in the following order:
    a. Read CRM Accounts
    b. Iterate CRM Accounts and Run Pipelines

7.1.9 Add Pipeline Batch

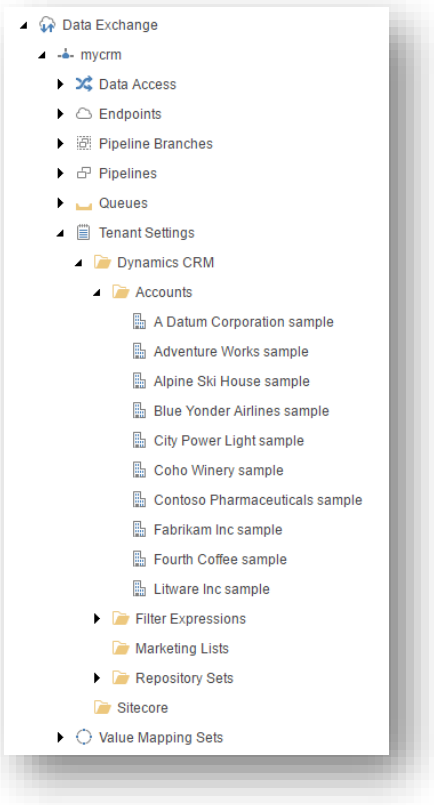
In order to be able to run the Pipelines, a Pipeline Batch is needed.

1. Navigate to the tenant "mycrm".
2. Navigate to `Pipeline Batches`.
3. Add the following item:
    a. **Template**: Pipeline Batch
    b. **Name**: CRM Accounts to xDB Sync Pipeline Batch
4. Navigate to `CRM Accounts to xDB Sync Pipeline Batch`.
5. Set the following field value:
    a. **Field name**: Pipelines
    b. **Value**: Read CRM Accounts Pipeline

6. Save the item.

7.1.10 Run Pipeline Batch

If you run the Pipeline Batch you should see the active CRM accounts as children under `/sitecore/system/Data Exchange/mycrm/Tenant Settings/Dynamics CRM/Accounts`.



7.2 Complex Synchronization

In section 7.1, you configured a synchronization process that represents CRM accounts as Sitecore items. The real power of having access to CRM accounts is when you are able to associate an xDB contact with a CRM account.

Some custom development is required. By default, Sitecore does not provide a way to store account information and to associate an xDB contact with the account. And since Sitecore does not store this information, Sitecore is not aware of this information.

Custom development is required for the following parts of the solution:

- Read the CRM contacts that are associated with an account from CRM
- Associate an xDB contact with an account
- Search for xDB contacts by account

In order to keep the instructions as simple as possible, a specific example is used to describe this configuration:

- This example builds on the example covered in section 7.1.
- Each CRM contact is a part of an account. You want to associate the Sitecore contact with the CRM account by setting a value on a custom contact facet.

Note
The relationship between contacts and accounts is similar to the relationship between contacts and marketing lists. The difference is that a contact is associated with a single account, while a contact may be associated with multiple marketing lists.

7.2.1 Extend Dynamics CRM Contact Facet

The default contact facet used to store CRM data on an xDB contact does not have the ability to store any account information. However, you can extend the contact facet so that it can.

Note
You can also create an entirely new contact facet, if you prefer. But since only a single value is going to be stored, there is not much value in creating a new contact facet.

1. In Visual Studio, create a new project:
    a. **Project template**: Class Library
    b. **Name**: Examples.DynamicsCrm
2. Add the following references to the project:
    a. Sitecore.Analytics.DynamicsCrm.dll
    b. Sitecore.Analytics.Model.dll

3. Add the following interface:

```
using Sitecore.Analytics.DynamicsCrm.Models;
using System;
namespace Examples.DynamicsCrm.Models
{
    public interface ICrmContactDataEx : ICrmContactData
    {
        Guid AccountId { get; set; }
    }
}
```

4. Add the following class:

```
using Sitecore.Analytics.DynamicsCrm.Models;
using System;

namespace Examples.DynamicsCrm.Models
{
    public class CrmContactDataEx : CrmContactData, ICrmContactDataEx
    {
        public CrmContactDataEx()
        {
            base.EnsureAttribute<Guid>(nameof(AccountId));
        }
        public Guid AccountId
        {
            get
            {
                return base.GetAttribute<Guid>(nameof(AccountId));
            }

            set
            {
                base.SetAttribute<Guid>(nameof(AccountId), value);
            }
        }
    }
}
```

7.2.2 Extend Indexable Contact

The contact facet makes it possible to store the account ID on the xDB contact, but in order for this value to be usable within Sitecore, the account ID must be indexed.

Sitecore includes the type `Sitecore.ContentSearch.Analytics.Models.ContactIndexable`. This type converts an xDB contact into a format where the contact data can be indexed.

Dynamics CRM Connect extends this type so that the properties on the `Sitecore.Analytics.DynamicsCrm.Models.ICrmContactData` contact facet are indexed. This is the purpose of the type `Sitecore.Analytics.DynamicsCrm.Search.ContactIndexableEx`.

You must extend `ContactIndexableEx` to include the account ID.

5. In Visual Studio, add the following references to the project:
    a. Sitecore.ContentSearch.dll
    b. Sitecore.ContentSearch.Analytics.dll

6. Add the following class:

```
using Examples.DynamicsCrm.Models;
using Sitecore.Analytics.DynamicsCrm.Search;
using Sitecore.Analytics.Model.Entities;
using System;

namespace Examples.DynamicsCrm.Search
{
    public class ContactIndexableEx2 : ContactIndexableEx
    {
        public ContactIndexableEx2(IContact contact) : base(contact)
        {
            LoadFields(contact);
        }
        private void LoadFields(IContact contact)
        {
            var data = contact.GetFacet<ICrmContactDataEx>("DynamicsCrm");
            if (data == null)
            {
                return;
            }
            AddField<Guid>("crm.account", data.AccountId);
        }
    }
}
```

7.2.3 Extend Contact Aggregator

When an xDB contact is changed, the aggregation process runs. The aggregation process ensures the contact is reindexed.

A component called a contact aggregator is responsible for creating the indexable contact from the xDB contact. In other words, the contact aggregator converts the xDB contact into an instance of the class you created in section 7.2.2.

You must create a custom contact aggregator that uses the indexable contact class.

7. In Visual Studio, add the following references to the project:
    a. Sitecore.Analytics.Aggregation.dll
    b. Sitecore.Kernel.dll

8. Add the following class:

```
        using Examples.DynamicsCrm.Search;
        using Sitecore.Analytics.Aggregation.Pipelines.ContactProcessing;
        using Sitecore.Analytics.DynamicsCrm.Aggregators;
        using Sitecore.ContentSearch;
        using Sitecore.ContentSearch.Analytics.Models;
        using System;
        using System.Collections.Generic;

        namespace Examples.DynamicsCrm.Aggregators
        {
            public class ContactChangeContactAggregatorEx2 : ContactChangeContactAggregatorEx
            {
                public ContactChangeContactAggregatorEx2(string name) : base(name) { }
                protected override IEnumerable<IContactIndexable>
ResolveIndexables(ContactProcessingArgs args)
                {
                    if (args == null)
                    {
                        throw new ArgumentNullException("args");
                    }
                    var changeEventReason = this.GetChangeEventReason(args);
                    var contact = args.GetContact();
                    if (contact != null)
                    {
                        yield return new ContactChangeIndexable(new
ContactIndexableEx2(contact), changeEventReason);
                    }
                    else
                    {
                        yield return new ContactChangeIndexable(new
IndexableUniqueId<Guid>(args.ContactId.Guid), changeEventReason);
                    }
                }
            }
        }
```

9. Compile the project.
10. Deploy Examples.DynamicsCrm.dll to your Sitecore server.

7.2.4 Register Contact Facet & Aggregator

You must change the configuration in Sitecore to use the new contact facet model and contact aggregator. This configuration will also ensure the new value on the contact facet is indexed properly.

1. Add the following file:
    a. **Path**: App_Config\Include\DynamicsCrm
    b. **File name**: xExamples.DynamicsCrm.config

Note
Config file are processed in alphabetical order. The file name starts with "x" in order to ensure it is processed after the file `Sitecore.Analytics.DynamicsCrm.config`.

Note
The configuration included in these instructions is for the Lucene search engine. If you are using Solr you must change the node /`configuration/sitecore/contentSearch/configuration`.

2. Add the following configuration to the file. This will replace the default contact facet model with the new, extended model, and the default contact aggregator with the new, extended aggregator:

```
        <configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
          <sitecore>

            <model>
              <elements>
                <element interface="Sitecore.Analytics.DynamicsCrm.Models.ICrmContactData,
Sitecore.Analytics.DynamicsCrm"
```

```
implementation="Sitecore.Analytics.DynamicsCrm.Models.CrmContactData,
Sitecore.Analytics.DynamicsCrm">
                  <patch:attribute
name="interface">Examples.DynamicsCrm.Models.ICrmContactDataEx,
Examples.DynamicsCrm</patch:attribute>
                  <patch:attribute
name="implementation">Examples.DynamicsCrm.Models.CrmContactDataEx,
Examples.DynamicsCrm</patch:attribute>
                </element>
              </elements>
              <entities>
                <contact>
                  <facets>
                    <facet name="DynamicsCrm"
contract="Sitecore.Analytics.DynamicsCrm.Models.ICrmContactData,
Sitecore.Analytics.DynamicsCrm">
                      <patch:attribute
name="contract">Examples.DynamicsCrm.Models.ICrmContactDataEx,
Examples.DynamicsCrm</patch:attribute>
                    </facet>
                  </facets>
                </contact>
              </entities>
            </model>

            <pipelines>
              <group groupName="analytics.aggregation">
                <pipelines>
                  <contacts>
                    <processor
type="Sitecore.Analytics.DynamicsCrm.Aggregators.ContactChangeContactAggregatorEx,
Sitecore.Analytics.DynamicsCrm">
                      <patch:attribute
name="type">Examples.DynamicsCrm.Aggregators.ContactChangeContactAggregatorEx2,
Examples.DynamicsCrm</patch:attribute>
                    </processor>
                  </contacts>
                </pipelines>
              </group>
            </pipelines>

            <contentSearch>
              <configuration type="Sitecore.ContentSearch.ContentSearchConfiguration,
Sitecore.ContentSearch">
                <indexes hint="list:AddIndex">
                  <index id="sitecore_analytics_index"
type="Sitecore.ContentSearch.LuceneProvider.LuceneIndex,
Sitecore.ContentSearch.LuceneProvider">
                    <param desc="name">$(id)</param>
                    <param desc="folder">$(id)</param>
                    <param desc="propertyStore"
ref="contentSearch/indexConfigurations/databasePropertyStore" param1="$(id)" />
                    <configuration
ref="contentSearch/indexConfigurations/defaultLuceneIndexConfiguration">
                      <fieldMap
ref="contentSearch/indexConfigurations/defaultLuceneIndexConfiguration/fieldMap">
                        <fieldNames hint="raw:AddFieldByFieldName">
                          <field fieldName="crm.account" storageType="YES"
indexType="TOKENIZED" vectorType="NO" boost="1f" type="System.Guid"
settingType="Sitecore.ContentSearch.LuceneProvider.LuceneSearchFieldConfiguration,
Sitecore.ContentSearch.LuceneProvider">
                            <analyzer
type="Sitecore.ContentSearch.LuceneProvider.Analyzers.LowerCaseKeywordAnalyzer,
Sitecore.ContentSearch.LuceneProvider"/>
                          </field>
                        </fieldNames>
                      </fieldMap>
                    </configuration>
                  </index>
                </indexes>
              </configuration>
            </contentSearch>

          </sitecore>
        </configuration>
```

7.2.5 Implement Entity Repository for Account Membership

The default entity repository is able to read entities from Dynamics CRM. If you are only reading attributes from the entity, the default entity repository can be used. But in cases where you need to read data from multiple entities, you need to create a custom entity repository.

For this example, you need an entity repository that is able to read the contacts associated with an account.

1. In Visual Studio, add the following references to the project:
    a. Sitecore.Connect.Crm.dll
    b. Sitecore.Connect.Crm.DynamicsCrm.dll
    c. Sitecore.DataExchange.dll
    d. Sitecore.DataExchange.Providers.DynamicsCrm.dll
    e. Sitecore.Services.Core.dll
2. Add a reference to the following NuGet packages:
    a. Microsoft.CrmSdk.CoreAssemblies (version 7.1.1)
3. Add the following class:

```
        using Microsoft.Xrm.Sdk;
        using Microsoft.Xrm.Sdk.Query;
        using Sitecore.Connect.Crm.DynamicsCrm.Extensions;
        using Sitecore.Connect.Crm.DynamicsCrm.Repositories;
        using Sitecore.Connect.Crm.Repositories;
        using System;
        using System.Collections.Generic;

        namespace Examples.DynamicsCrm.Repositories
        {
            public class XrmClientAccountMembershipRepository :
BaseXrmClientMemberEntitiesRepository
            {
                public XrmClientAccountMembershipRepository(string entityName) :
base(entityName)
                {
                }
                protected override IEnumerable<Entity> ReadMembers(Guid entityId,
RepositoryOperationContext context)
                {
                    var query = new QueryExpression("contact");
                    var settings = context.GetReadEntityRepositoryOperationSettings();
                    if (settings != null)
                    {
                        query.ColumnSet = settings.ColumnSet;
                    }
                    var accountEntity = new LinkEntity("contact", "account",
"parentcustomerid", "accountid", JoinOperator.Inner);
                    accountEntity.Columns = new ColumnSet("accountid", "name");
                    accountEntity.EntityAlias = "account";
                    query.LinkEntities.Add(accountEntity);
                    query.Criteria.Conditions.Add(new ConditionExpression("parentcustomerid",
ConditionOperator.Equal, entityId));
                    return ReadEntities(query, context);
                }
            }
        }
```

4. Compile the project.
5. Deploy Examples.DynamicsCrm.dll to your Sitecore server.

7.2.6 Add Template for Entity Repository

A template is needed in order to configure the entity repository.

1. In Sitecore, open Template Manager.
2. Navigate to `Templates/Data Exchange/Providers/Dynamics CRM/Repositories`.
3. Add a new template named "XRM Client Account Membership Repository".
4. Set the icon for this template to `Office/32x32/data.png`.
5. Add the template `Templates/Data Exchange/Providers/Dynamics CRM/Repositories/XRM Client Entity Repository` as a base template.

7.2.7 Implement Converter for Entity Repository Template

A converter is needed to transform items created using the template from section 7.2.6 into entity repository objects that can be used by Pipeline Step Processors.

1. In Visual Studio, add the following class:

```
        using Examples.DynamicsCrm.Repositories;
        using Sitecore.Connect.Crm.DynamicsCrm.Repositories;
        using Sitecore.DataExchange.Converters;
        using Sitecore.DataExchange.Providers.DynamicsCrm.Models.ItemModels.Repositories;
        using Sitecore.DataExchange.Repositories;
        using Sitecore.Services.Core.Model;
        using System;

        namespace Examples.DynamicsCrm.Converters
        {
            public class XrmClientAccountMembershipRepositoryConverter :
BaseItemModelConverter<ItemModel, XrmClientEntityRepository>
            {
                private static readonly Guid TemplateId = Guid.Parse("[TEMPLATE-ID]");
                public XrmClientAccountMembershipRepositoryConverter(IItemModelRepository
repository) : base(repository)
                {
                    this.SupportedTemplateIds.Add(TemplateId);
                }
                public override XrmClientEntityRepository Convert(ItemModel source)
                {
                    if (!CanConvert(source))
                    {
                        return null;
                    }
                    var repository = new
XrmClientAccountMembershipRepository(base.GetStringValue(source,
EntityRepositoryItemModel.EntityName));
                    return repository;
                }
            }
        }
```

2. Find the following line in the code:

```
        private static readonly Guid TemplateId = Guid.Parse("[TEMPLATE-ID]");
```

3. Replace [TEMPLATE-ID] with the ID for the template from section 7.2.6.
4. Compile the project.
5. Deploy Examples.DynamicsCrm.dll to your Sitecore server.

7.2.8 Assign Converter to Entity Repository Template

The converter created in section 7.2.7 is always used to convert items based on the template created in section 7.2.6. This can be set up on the template so the converter is assigned automatically when a new item is created.

1. In Sitecore, open Template Manager.
2. Navigate to `Templates/Data Exchange/Providers/Dynamics CRM/Repositories/XRM Client Account Membership Repository`.
3. Add standard values to the template.
4. Navigate to the standard values item.
5. Set the following field value:
    a. **Field name**: Entity name
    b. **Value**: accountmembership
6. Set the following field value:
    a. **Field name**: Converter type
    b. **Value**: `Examples.DynamicsCrm.Converters.XrmClientAccountMembershipRepositoryConverter, Examples.DynamicsCrm`
7. Save the item.

7.2.9 Add Template for Read Members Pipeline Step

A template is needed in order to configure the Pipeline Step that reads members from a CRM account using the entity repository.

1. In Sitecore, open Template Manager.
2. Navigate to `Templates/Data Exchange/Providers/Dynamics CRM/Pipeline Steps`.
3. Add a new template named "Read CRM Account Membership Pipeline Step".
4. Set the icon for this template to `Office/32x32/element.png`.
5. Add the template `Templates/Data Exchange/Providers/Dynamics CRM/Pipeline Steps/Read CRM Entities Pipeline Step` as a base template.

7.2.10 Implement Converter for Read Members Pipeline Step

A converter is needed to transform items created using the template from section 7.2.9 into entity repository objects that can be used by Pipeline Step Processors.

1. In Visual Studio, add the following class:

```
        using Sitecore.DataExchange.Providers.DynamicsCrm.Converters.PipelineSteps;
        using Sitecore.DataExchange.Repositories;
        using System;

        namespace Examples.DynamicsCrm.Converters
        {
            public class ReadAccountMembershipStepConverter : ReadEntitiesStepConverter
            {
                private static readonly Guid TemplateId = Guid.Parse("[TEMPLATE-ID]");
                public ReadAccountMembershipStepConverter(IItemModelRepository repository) :
base(repository)
                {
                    this.SupportedTemplateIds.Add(TemplateId);
                }
            }
        }
```

2. Find the following line in the code:

```
        private static readonly Guid TemplateId = Guid.Parse("[TEMPLATE-ID]");
```

3. Replace [TEMPLATE-ID] with the ID for the template from section 7.2.9.
4. Compile the project.
5. Deploy Examples.DynamicsCrm.dll to your Sitecore server.

7.2.11 Implement Processor for Read Members Pipeline Step

A processor implements the logic for the Pipeline Step. A processor is needed to read members from a CRM account using the Entity Repository.

1. In Visual Studio, add the following class:

```
        using Microsoft.Xrm.Sdk;
        using Sitecore.Connect.Crm.DynamicsCrm.Plugins;
        using Sitecore.Connect.Crm.Repositories;
        using Sitecore.DataExchange.Contexts;
        using Sitecore.DataExchange.Extensions;
        using Sitecore.DataExchange.Models;
        using Sitecore.DataExchange.Providers.DynamicsCrm.Processors.PipelineSteps;
        using System;

        namespace Examples.DynamicsCrm.Processors
        {
            public class ReadAccountMembershipStepProcessor : ReadEntitiesStepProcessor
            {
                protected override RepositoryOperationContext
GetRepositoryOperationContext(Endpoint endpoint, PipelineStep pipelineStep, PipelineContext
pipelineContext)
                {
                    var opContext = base.GetRepositoryOperationContext(endpoint, pipelineStep,
pipelineContext);
                    if (opContext == null)
                    {
                        return null;
                    }
                    var syncSettings = pipelineContext.GetSynchronizationSettings();
                    if (syncSettings == null)
                    {
                        return null;
                    }
                    var source = syncSettings.Source;
                    if (source == null)
                    {
                        return null;
                    }
                    var entity = source as Entity;
                    if (entity == null)
                    {
                        return null;
                    }
                    if (!entity.Attributes.ContainsKey("accountid"))
                    {
                        return null;
                    }
                    var accountId = (Guid)entity.Attributes["accountid"];
                    if (accountId == Guid.Empty)
                    {
                        return null;
                    }
                    var settings = new MemberEntitiesSettings
                    {
                        EntityId = accountId
                    };
                    opContext.Plugins.Add(settings);
                    return opContext;
                }
            }
        }
```

2. Compile the project.
3. Deploy Examples.DynamicsCrm.dll to your Sitecore server.

7.2.12 Assign Converter & Processor to Pipeline Step Template

The converter created in section 7.2.10 is always used to convert items based on the template created in section 7.2.9. The processor created in section is always used to run the Pipeline Step. This can be set up on the template so the converter is assigned automatically when a new item is created.

1. In Sitecore, open Template Manager.
2. Navigate to `Templates/Data Exchange/Providers/Dynamics CRM/Pipeline Steps/Read CRM Account Membership Pipeline Step`.
3. Add standard values to the template.
4. Navigate to the standard values item.
5. Set the following field value:
    a. **Field name**: Converter type
    b. **Value**: `Examples.DynamicsCrm.Converters.ReadAccountMembershipStepConverter, Examples.DynamicsCrm`
6. Set the following field value:
    a. **Field name**: Processor type
    b. **Value**: `Examples.DynamicsCrm.Processors.ReadAccountMembershipStepProcessor, Examples.DynamicsCrm`
7. Save the item.

7.2.13 Add Entity Repository to Entity Repository Set

The entity repository must be added to the entity repository set.

1. In Sitecore, open Content Editor.
2. Navigate to the tenant "mycrm".
3. Navigate to `Tenant Settings/Dynamics CRM/Repository Sets/mycrm entity repository set`.
4. Add the following item:
    a. **Template**: XRM Client Account Membership Entity Repository
    b. **Name**: accountmembership

7.2.14 Add Value Accessor Set for CRM Account

In order to be able to read attributes from a CRM accountmembership entity, a Value Accessor Set is needed. This item identifies the attributes from the CRM accountmembership that are available to Dynamics CRM Connect.

1. Navigate to the tenant "mycrm".
2. Navigate to `Data Access/Value Accessor Sets/Providers/Dynamics CRM`.
3. Add the following item:
    a. **Template**: Entity Value Accessor Set
    b. **Name**: CRM Account Membership Attributes
4. Navigate to `CRM Account Membership Attributes`.

5. Add the following item:
    a. **Template**: Entity Attribute Value Accessor
    b. **Name**: Account Id
6. Set the following field value:
    a. **Field name**: Use value property
    b. **Value**: ticked
7. Set the following field value:
    a. **Field name**: Attribute name
    b. **Value**: account.accountid
8. Save the item.
9. Navigate to `CRM Account Membership Attributes`.
10. Add the following item:
    a. **Template**: Entity Attribute Value Accessor
    b. **Name**: Account Name
11. Set the following field value:
    a. **Field name**: Use value property
    b. **Value**: ticked
12. Set the following field value:
    a. **Field name**: Attribute name
    b. **Value**: account.accountname
13. Save the item.

7.2.15 Add Value Accessor Set for Sitecore Contact

In order to be able to write values to the Sitecore contact, a Value Accessor Set is needed. This item identifies the fields from the Sitecore contact that are available to Dynamics CRM Connect.

1. Navigate to the tenant "mycrm".
2. Navigate to `Data Access/Value Accessor Sets/Providers/Sitecore`.
3. Add the following item:
    a. **Template**: xDB Contact Value Accessor Set
    b. **Name**: xDB Contact Properties for Account Membership
4. Navigate to `xDB Contact Properties for Account Membership`.
5. Add the following item:
    a. **Template**: xDB Contact Facet Property Value Accessor
    b. **Name**: Account Id
6. Set the following field value:
    a. **Field name**: Contact facet name
    b. **Value**: DynamicsCrm

7. Set the following field value:
    a. **Field name**: Property name
    b. **Value**: AccountId
8. Save the item.

7.2.16 Add Value Mapping Set for Account Membership

In order to map values from the source object (CRM account membership entity) to a target object (Sitecore contact), you must define a Value Mapping Set.

1. Navigate to the tenant "mycrm".
2. Navigate to `Value Mapping Sets`.
3. Add the following item:
    a. **Template**: Value Mapping Set
    b. **Name**: CRM Account Membership to xDB Contact
4. Navigate to `CRM Account Membership to xDB Contact`
5. Add the following item:
    a. **Template**: Value Mapping
    b. **Name**: Account Id
6. Set the following field value:
    a. **Field name**: Value accessor for source object
    b. **Value**: Data Access/Value Accessor Sets/Providers/Dynamics CRM/CRM Account Membership Attributes/Account Id
7. Set the following field value:
    a. **Field name**: Value accessor for target object
    b. **Value**: Data Access/Value Accessor Sets/Providers/Sitecore/xDB Contact Properties for Account Membership/Account Id
8. Set the following field value:
    a. **Field name**: Transformer for source value
    b. **Value**: Value Readers/Common/Guid Value Reader
9. Save the item.

7.2.17 Add Pipelines to Handle Single CRM Account

Each account membership entity read from CRM is mapped to a Sitecore contact. A Pipeline is used to define the logic that resolves the Sitecore contact and applies data from the CRM entity.

1. Navigate to the tenant "mycrm".
2. Navigate to `Pipelines/CRM Account Pipelines`.
3. Add the following item:
    a. **Template**: Pipeline
    b. **Name**: CRM Account Membership to xDB Contact Sync Pipeline
4. Navigate to `CRM Account Membership to xDB Contact Sync Pipeline`.

5. Add the following item:
    a. **Template**: Resolve xDB Contact from Repository Pipeline Step
    b. **Name**: Resolve xDB Contact from Repository
6. Set the following field value:
    a. **Field name**: Endpoint to read data from
    b. **Value**: Sitecore/Local xDB Contacts Endpoint
7. Set the following field value:
    a. **Field name**: Identifier value accessor
    b. **Value**: Value Accessor Sets/Providers/Dynamics CRM/CRM Contact Attributes/Email
8. Set the following field value:
    a. **Field name**: Identifier object location
    b. **Value**: Pipeline Processor Context Source
9. Set the following field value:
    a. **Field name**: Resolved object location
    b. **Value**: Pipeline Processor Context Target
10. Save the item.
11. Navigate to `CRM Account Membership to xDB Contact Sync Pipeline`.
12. Add the following item:
    a. **Template**: Resolve xDB Contact from Queue Pipeline Step
    b. **Name**: Resolve xDB Contact from Queue
13. Set the following field value:
    a. **Field name**: Endpoint to read data from
    b. **Value**: Endpoints/Common/Queue Endpoint
14. Set the following field value:
    a. **Field name**: Identifier value accessor
    b. **Value**: Value Accessor Sets/Providers/Sitecore/xDB Contact Properties/xDB Contact Identifier
15. Set the following field value:
    a. **Field name**: Identifier object location
    b. **Value**: Pipeline Processor Context Target
16. Set the following field value:
    a. **Field name**: Resolved object location
    b. **Value**: Pipeline Processor Context Target
17. Save the item.
18. Navigate to `CRM Account Membership to xDB Contact Sync Pipeline`.
19. Add the following item:
    a. **Template**: Apply Mapping Pipeline Step
    b. **Name**: Apply Mapping

20. Set the following field value:
    a. **Field name**: Mapping set
    b. **Value**: Value Mapping Sets/CRM Account Membership to xDB Contact
21. Save the item.
22. Navigate to `CRM Account Membership to xDB Contact Sync Pipeline`.
23. Change the order of the child items to the following:
    a. Resolve xDB Contact from Repository
    b. Resolve xDB Contact from Queue
    c. Apply Mapping

7.2.18 Add Pipelines to Handle CRM Accounts

For each CRM account you must read the CRM contacts who are associated with the CRM account. A Pipeline is used to define the logic that reads the CRM contacts that are associated with the CRM account and iterates those CRM contacts.

Note
You have not yet configured the Pipeline that will read the entities from CRM. The Pipeline that handles each entity is used in the Pipeline that reads the entities, so the Pipeline that handles each entity is configured first.

1. Navigate to the tenant "mycrm".
2. Navigate to `Pipelines/CRM Account Pipelines`.
3. Add the following item:
    a. **Template**: Pipeline
    b. **Name**: Read Members from Single CRM Account Pipeline
4. Navigate to `Read Members from Single CRM Account Pipeline`.
5. Add the following item:
    a. **Template**: Read CRM Account Membership Pipeline Step
    b. **Name**: Read Members from Single CRM Account
6. Set the following field value:
    a. **Field name**: Endpoint for Dynamics CRM entity repository
    b. **Value**: Dynamics CRM/Dynamics CRM Entity Endpoint
7. Set the following field value:
    a. **Field name**: Entity name
    b. **Value**: accountmembership
8. Set the following field value:
    a. **Field name**: Attributes to read
    b. **Value**: Common CRM Entity Attributes, CRM Contact Attributes Minimal
9. Save the item.
10. Navigate to `Read Members from Single CRM Account Pipeline`.

11. Add the following item:
    a. **Template**: Iterate Data and Run Pipelines Pipeline Step
    b. **Name**: Iterate CRM Members and Run Pipelines
12. Set the following field value:
    a. **Field name**: Pipelines
    b. **Value**: CRM Account Membership to xDB Contact Sync Pipeline
13. Save the item.
14. Navigate to `Read Members from Single CRM Account Pipeline`.
15. Change the order of the child items to the following:
    a. Read Members from Single CRM Account
    b. Iterate CRM Members and Run Pipelines

7.2.19 Update Pipeline that Reads CRM Accounts

Currently, the pipeline that reads accounts from CRM is only creating or updating the Sitecore items that correspond to those accounts. That pipeline must be changed so that it reads the account membership from CRM.

1. Navigate to the tenant "mycrm".
2. Navigate to `Pipelines/CRM Account Pipelines/Read CRM Accounts Pipeline/Iterate CRM Accounts and Run Pipelines`.
3. In the field `Pipelines`, add the pipeline `Pipelines/CRM Account Pipelines/Read Members from Single CRM Account Pipeline`.
4. Set the order for the items in the field to the following:
    a. CRM Account to Sitecore Account Sync Pipeline
    b. Read Members from Single CRM Account Pipeline
5. Save the item.

7.2.20 Update Pipeline Batch

Now that entries are being added to the work queue, the Pipeline Batch must be updated to processes the queue.

1. Navigate to the tenant "mycrm".
2. Navigate to `Pipeline Batches/CRM Accounts to xDB Sync Pipeline Batch`.
3. In the field `Pipelines`, add the pipeline `Pipelines/Process Contacts in Queue Pipeline`.
4. Set the order for the items in the field to the following:
    a. Read CRM Accounts Pipeline
    b. Process Contacts in Queue Pipeline
5. Save the item.

7.2.21 Run Pipeline Batch

If you run the Pipeline Batch you should see the CRM account ID included in the Sitecore contact. The value will appear as a property "AccountId" on the "DynamicsCrm" contact facet.

7.2.22 Add to Full CRM Contacts to xDB Sync Pipeline Batch

If you want, you can add the CRM account synchronization process to the process that synchronizes CRM contact and marketing list data.

1. Navigate to the tenant "mycrm".
2. Navigate to `Pipeline Batches/Full CRM Contacts to xDB Sync Pipeline Batch`.
3. In the field `Pipelines`, add the pipeline `Pipelines/CRM Account Pipelines/Read CRM Accounts Pipeline`.
4. Set the order for the items in the field to the following:
    a. Reset Existing xDB Contacts Pipeline
    b. Read CRM Lists Pipeline
    c. Read CRM Contacts Pipeline
    d. Read CRM Accounts Pipeline
    e. Process Contacts in Queue Pipeline
5. Save the item.

If you run the Pipeline Batch you should see the CRM account ID included in the Sitecore contact.

7.3 Using Custom Entity Data

Getting data from CRM into Sitecore is only useful if the data can actually be used within Sitecore. There are many ways CRM data can be used within Sitecore. This section covers a couple of the possibilities.

7.3.1 Personalization

By adding a custom condition for the Sitecore Rules Editor you can allow content authors to personalize web pages using the visitor's account.

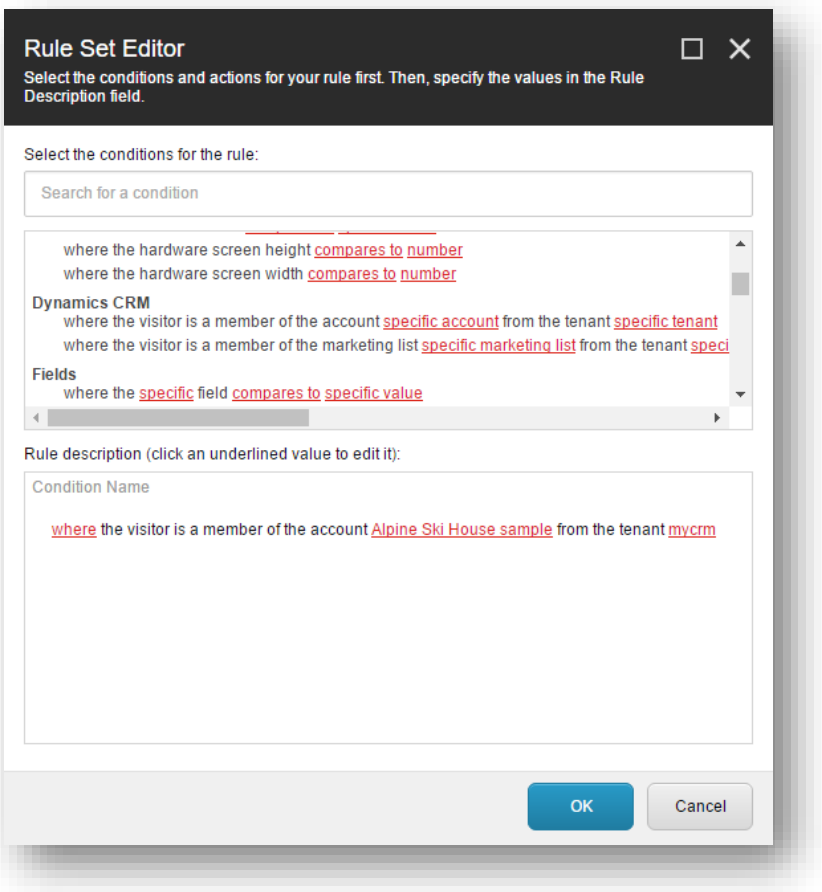
1. In Visual Studio, add the following references to the project:
    a. Sitecore.Analytics.dll
2. Add the following class:

```
        using Examples.DynamicsCrm.Models;
        using Sitecore.Analytics.DynamicsCrm.Rules.Contacts.Conditions;
        using Sitecore.Analytics.Tracking;
        using Sitecore.Rules;
        using System;

        namespace Examples.DynamicsCrm.Rules
        {
            public class PersonalizationAccountCondition<T> : BaseMembershipCondition<T> where
T : RuleContext
            {
                public PersonalizationAccountCondition() : base("AccountId")
                {
                }
                protected override bool DoesMatch(Guid entityId, Contact contact)
                {
                    var facet = contact.GetFacet<ICrmContactDataEx>(this.ContactFacetName);
                    if (facet == null)
                    {
                        return false;
                    }
                    return facet.AccountId == entityId;
                }
            }

        }
```

3. Compile the project.
4. Deploy Examples.DynamicsCrm.dll to your Sitecore server.
5. In Content Editor, navigate to `/sitecore/system/Settings/Rules/Definitions/Elements/Dynamics CRM Visitor`.
6. Add the following item:
    a. **Template**: Condition
    b. **Name**: Dynamics CRM Account
7. Set the following field value:
    a. **Field name**: Text
    b. **Value**: where the visitor is a member of the account [externalentityid,CRM Connect/DependentTree,dependency=tenant&mode=descendant&templateid={0DE319D9-125C-42F3-B330-05C30D1B42D3}&rootitemname=Accounts&selection=[ACCOUNT-TEMPLATE-ID]&setRootAsSearchRoot=true,specific account] from the tenant [tenant,Tree,root={5EE8330D-E35E-433B-9BA6-DAF87ED38867}&selection={327A381B-59F8-4E88-B331-BEBC7BD87E4E}&setRootAsSearchRoot=true,specific tenant]
8. In the field "Text", change "[ACCOUNT-TEMPLATE-ID]" to the ID of the template created in section 7.1.2.

9. Set the following field value:
    a. **Field name**: Type
    b. **Value**: Examples.DynamicsCrm.Rules.PersonalizationAccountCondition, Examples.DynamicsCrm
10. Save the item.

Now you can configure a personalization condition based on whether or not the visitor is a member of a specific CRM account.



7.3.2 Segmentation

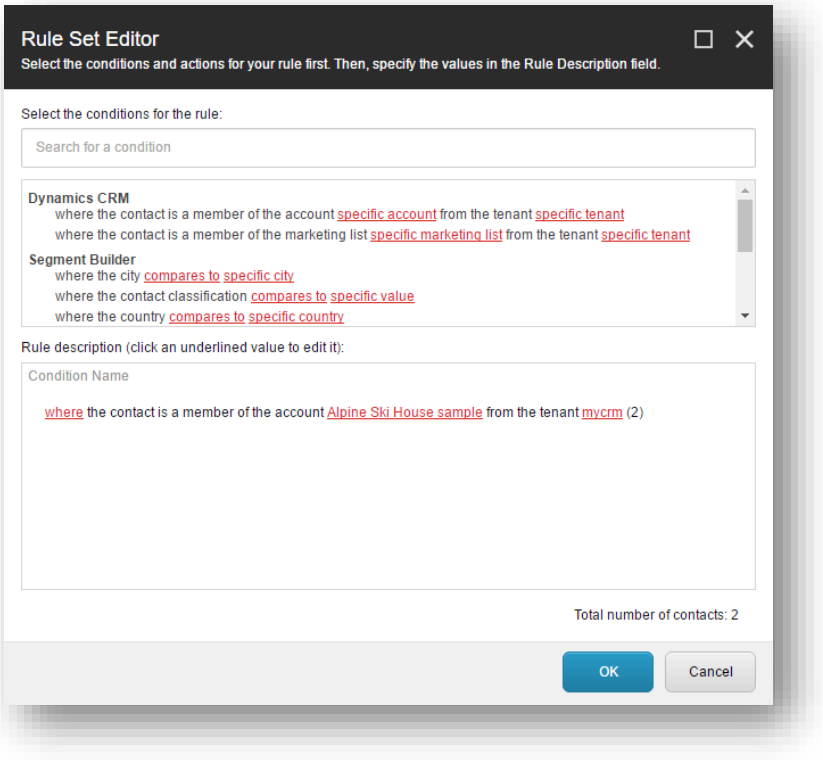
By adding a custom condition for the Sitecore Rules Editor, you can allow Sitecore List Manager users to use membership in a CRM account as a segmentation condition.

1. In Visual Studio, add the following references to the project:
    a. Sitecore.SegmentBuilder.dll
2. Add the following class:

```
        using Sitecore.Analytics.DynamicsCrm.Rules.SegmentBuilder.Conditions;
        using Sitecore.Analytics.Rules.SegmentBuilder;
        using Sitecore.ContentSearch.Analytics.Models;
        using System;

        namespace Examples.DynamicsCrm.Rules
        {
            public class SegmentationAccountCondition<T> : BaseMembershipCondition<T> where T
: VisitorRuleContext<IndexedContact>
            {
                public SegmentationAccountCondition() : base("AccountId", "crm.account")
                {
                }
            }
        }
```

3. Compile the project.
4. Deploy Examples.DynamicsCrm.dll to your Sitecore server.
5. In Content Editor, navigate to `/sitecore/system/Settings/Rules/Definitions/Elements/Dynamics CRM Segment Builder`.
6. Add the following item:
    a. **Template**: Condition
    b. **Name**: Dynamics CRM Account
7. Set the following field value:
    a. **Field name**: Text
    b. **Value**: where the contact is a member of the account [externalentityid,CRM Connect/DependentTree,dependency=tenant&mode=descendant&templateid={0DE319D9-125C-42F3-B330-05C30D1B42D3}&rootitemname=Accounts&selection=[ACCOUNT-TEMPLATE-ID]&setRootAsSearchRoot=true,specific account] from the tenant [tenant,Tree,root={5EE8330D-E35E-433B-9BA6-DAF87ED38867}&selection={327A381B-59F8-4E88-B331-BEBC7BD87E4E}&setRootAsSearchRoot=true,specific tenant]
8. In the field "Text", change "[ACCOUNT-TEMPLATE-ID]" to the ID of the template created in section 7.1.2.
9. Set the following field value:
    a. **Field name**: Type
    b. **Value**: Examples.DynamicsCrm.Rules.SegmentationAccountCondition, Examples.DynamicsCrm
10. Save the item.

Now you can configure a personalization condition based on whether or not the visitor is a member of a specific CRM account.



Chapter 8

Advanced Usage

This chapter covers advanced usage scenarios for Dynamics CRM Connect.

8.1 Tenants

A Tenant is a very simple object: it is used to organize and isolate settings so that multiple configurations can coexist. This section describes configuration options for Tenants.

8.1.1 Renaming a Tenant

Within xDB, certain contact facets support multiple values. For example, a contact may have multiple email addresses. Each of these email addresses has an identifier or name associated with it, such as "work" or "home".

By default, when these sorts of values are read from the CRM, the name of the Tenant is used as the identifier. If the name of the Tenant is "mycrm", when you synchronize a CRM contact, the email address that is read from the CRM will be the "mycrm" email address.

1. In Content Editor, navigate to the Tenant you want to rename.
2. Rename the Sitecore item.
3. Navigate to `Data Access/Value Readers/Providers/Dynamics CRM/Constant Value Reader - Tenant Name`.
4. In the field `Value`, enter the new Tenant name.
5. Save the item.
6. Navigate to `Data Access/Value Accessor Sets/Providers/Sitecore/xDB Contact Properties`.
7. For each item that uses the template `xDB Contact Facet Indexer Property Value Accessor`, change the value of the property `Index name for indexer property` to match the new tenant name.

Note
Renaming a Tenant will not change data that already exists in xDB. The next time data is synchronized, the new Tenant name will be used. However, this may result in duplicate data in xDB.

For example, if you have already run the contact synchronization process and the Tenant name is "mycrm", you will have an email address identified as the "mycrrm" email address. If you rename the Tenant to "Production CRM" and run the process again, you will not have an email address identified as the "Production CRM" email address. The "mycrm" email address will remain.

8.1.2 Securing a Tenant

Since a Tenant is defined as a Sitecore item, a Tenant can be secured the same way any other Sitecore item is secured: via Sitecore authorizations.

8.2 Ad Hoc Synchronization

Up to now, most of the information in this document involves synchronizing multiple entities asynchronously. However, it is possible to use Dynamics CRM Connect to synchronize a specific entity synchronously.

This section describes how to trigger the synchronization of data from a contact in xDB to a contact in CRM. This could be used, for example, to automatically update CRM when a visitor's session ends on the website.

Note
While this section covers a specific scenario, the same logic can be used to synchronize any data.

8.2.1 Determine the Appropriate Pipeline

Running a synchronization process normally involves running a Pipeline Batch. But Pipeline Batches are designed to handle sets of data, and to be run asynchronously.

In this case, the synchronization only handles a single object, and it needs to run synchronously. Specifically, it needs to handle a single xDB contact.

The Pipeline Batch "xDB Contacts to CRM Sync Pipeline Batch" contains this logic. By looking at the Pipelines that this Pipeline Batch runs, you can see that the Pipeline "Upsert Single CRM Contact Pipeline" is responsible for handling a single xDB contact. This is the Pipeline you are looking for.

Note the item ID for the Pipeline definition item.

8.2.2 Determine the Required Context Values

After you have determined which Pipeline needs to run, you must determine the context values that the steps in the Pipeline are expecting.

For Pipelines included in Dynamics CRM Connect, this is documented in the detailed sequence diagrams available with the product documentation. If you are working with a custom Pipeline, you will need to determine this for yourself.

The steps in the Pipeline "Upsert Single CRM Contact Pipeline" expect the following values be set:

- Source object is set to the xDB contact

8.2.3 Run the Pipeline

The following code demonstrates how to run the Pipeline "Upsert Single CRM Contact Pipeline".

Note
The code assumes you have already resolved the xDB contact you want to use to update CRM. This contact is accessed through the variable `contact`.

Note
The code uses extension methods defined in the namespace `Sitecore.DataExchange.Extensions`, so you must import this namespace in your code.

```
        Contact contact = ... //contact comes from the tracker or from somewhere else
        Guid pipelineItemId = Guid.Parse("####"); //item id for the pipeline definition item
        IItemModelRepository itemModelRepo =
Sitecore.DataExchange.Context.ItemModelRepository;
        ItemModel model = itemModelRepo.Get(pipelineItemId);
        IConverter<ItemModel, Pipeline> converter =
model.GetConverter<Pipeline>(itemModelRepo);
        Pipeline pipeline = converter.Convert(model);
        IPipelineProcessor processor = pipeline.PipelineProcessor;
        PipelineBatchContext batchContext = new PipelineBatchContext();
        PipelineContext pipelineContext = new PipelineContext(batchContext);
        pipelineContext.Plugins.Add(new SynchronizationSettings { Source = contact });
        if (processor.CanProcess(pipeline, pipelineContext))
        {
          processor.Process(pipeline, pipelineContext);
          if (!pipelineContext.CriticalError)
          {
            //success
          }
        }
```

Chapter 9

Component Reference

Dynamics CRM Connect is a highly customizable product. You are able to change virtually all of the functionality that is provided out-of-the-box.

This chapter describes the various components that are available.

9.1 Filter Expressions for Dynamics CRM data

Filter expressions are used to limit the data that is read from Dynamics CRM. These expressions are used to build the query for Dynamics CRM. As such, they reflect options that are available from the Dynamics CRM API.

Note
It is important that you understand the implications of using filter expressions before you configure them. These expressions will change the data that is read from Dynamics CRM. You must consider that data may not be read from Dynamics CRM may already exist in Sitecore.

Consider the example of synchronizing marketing list membership. If you use a filter expression so that only contacts that have changed in the past 24 hours, is that set of contacts going to accurately reflect the marketing list membership?

9.1.1 Boolean Condition Expression

A Boolean condition expression is used to compare the value of an attribute to either `true` or `false`.

When configuring a Boolean condition expression, you must provide the following values:

Field	Description
Value accessor for entity attribute	The value accessor that is used to read the value of an attribute from an entity. It is important that the value accessor selected has a field that identifies the attribute name. The attribute name is used to build the filter for the API call to Dynamics CRM.
Is	The value that is used in the expression.

Note
Numeric condition expression items are converted into instances of the type `Microsoft.Xrm.Sdk.Query.ConditionExpression`. For more information about how this type works, see https://msdn.microsoft.com/en-us/library/gg334419.aspx.

9.1.2 Date Range Condition Expression

A date condition expression is used to determine whether or not the value of an attribute is within a date range.

When configuring a date condition expression, you must provide the following values:

Field	Description
Value accessor for entity attribute	The value accessor that is used to read the value of an attribute from an entity. It is important that the value accessor selected has a field that identifies the attribute name. The attribute name is used to build the filter for the API call to Dynamics CRM.
Start date	The start date for the expression. If no value is specified, no lower bound is considered.
End date	The end date for the expression. If no value is specified, no upper bound is considered.

Note
String condition expression items are converted into instances of the type `Microsoft.Xrm.Sdk.Query.ConditionExpression`. For more information about how this type works, see https://msdn.microsoft.com/en-us/library/gg334419.aspx.

9.1.3 Numeric Condition Expression

A numeric condition expression is used to compare the value of an attribute to a numeric value.

When configuring a numeric condition expression, you must provide the following values:

Field	Description
Value accessor for entity attribute	The value accessor that is used to read the value of an attribute from an entity. It is important that the value accessor selected has a field that identifies the attribute name. The attribute name is used to build the filter for the API call to Dynamics CRM.
Condition operator	The operator that is used in the expression.
Value	The integer value that is used in the expression.

Note
Numeric condition expression items are converted into instances of the type `Microsoft.Xrm.Sdk.Query.ConditionExpression`. For more information about how this type works, see https://msdn.microsoft.com/en-us/library/gg334419.aspx.

9.1.4 Relative Date Condition Expression

A relative date condition expression is used to compare the value of an attribute to a date that is relative to a known date.

The following are examples of relative date conditions:

- Within the last week
- Within the next 30 days

A relative date condition expression is implemented as a dynamically generated date range condition. To use the examples above:

Description	Start date	End date
Within the last week	Today - 7 days	Today
Within the next 30 days	Today	Today + 30 days

When configuring a relative date condition expression, you must provide the following values:

Field	Description
Value accessor for entity attribute	The value accessor that is used to read the value of an attribute from an entity. It is important that the value accessor selected has a field that identifies the attribute name. The attribute name is used to build the filter for the API call to Dynamics CRM.
Operator	This value is used to determine the start and end dates for the condition expression.
Offset	The integer value that is used to determine the start and end dates for the condition expression. When this is a negative number, a date in the past is used. When this is a positive number, a date in the future is used.
Unit of time	This value is used to determine the start and end dates for the condition expression.

Note
Relative condition expression items are converted into instances of the type `Microsoft.Xrm.Sdk.Query.ConditionExpression`. For more information about how this type works, see https://msdn.microsoft.com/en-us/library/gg334419.aspx.

9.1.5 String Condition Expression

A string condition expression is used to compare the value of an attribute to a string value.

When configuring a string condition expression, you must provide the following values:

Field	Description
Value accessor for entity attribute	The value accessor that is used to read the value of an attribute from an entity. It is important that the value accessor selected has a field that identifies the attribute name. The attribute name is used to build the filter for the API call to Dynamics CRM.
Condition operator	The operator that is used in the expression.
Value	The string value that is used in the expression.

Note
String condition expression items are converted into instances of the type `Microsoft.Xrm.Sdk.Query.ConditionExpression`. For more information about how this type works, see https://msdn.microsoft.com/en-us/library/gg334419.aspx.

9.1.6 Filter Expression

This is the "top-level" item for the definition of a filter expression. One or more expressions must be added as children.

The template has a single field: `Logical operator`. This value determines the relationship between the child expressions:

- `And` means that all of the child expressions must be true
- `Or` means that at least one of the child expressions must be true

Adding a filter expression as a child of another filter expression gives you the ability to build more complex expressions. For example, you can build an expression that includes contacts whose email address contains "example.com" or "example.org" and whose region is "Europe".

Note
Filter expression items are converted into instances of the type `Microsoft.Xrm.Sdk.Query.FilterExpression`. For more information about how this type works, see https://msdn.microsoft.com/en-us/library/gg309410.aspx.

9.2 Value Accessors for xDB Contact data

Dynamics CRM Connect includes Value Accessors that handle the most common values that are stored in contacts.

Note
Each Value Accessor template includes fields `Value reader` and `Value writer`. In general, these fields can be ignored.

The different Value Accessors use the other fields on the template to determine the appropriate Value Reader and Value Writer. The Value Reader and Value Writer are automatically assigned.

These fields are available in cases where you need to explicitly set a Value Reader and/or Value Writer (meaning the Value Reader or Value Writer that is automically determined is not appropriate in a specific case).

9.2.1 xDB Contact Facet Property Value Accessor

Most values stored in contact facets are stored in basic properties. This includes text, numbers and dates.

For example, values like the contact's first name, last name (surname) and job title are simple string values, so this type of Value Accessor is appropriate.

Field	Description
Contact facet name	Name of the contact facet to access.
Property name	Name of the property on the contact facet.

9.2.2 xDB Contact Facet Indexer Property Value Accessor

This Value Accessor is used for "dictionary properties". A dictionary property is a property on a contact facet that stores multiple values, where each value is identified by a key.

For example, email addresses are stored in a dictionary property, because a contact might have a work email address, home email address and a backup email address. In this example, the keys might be "work", "home" and "other".

Other values that are stored in dictionary properties are addresses and telephone numbers.

Field	Description
Contact facet name	Name of the contact facet to access.
Property name of indexer property	Name of the property on the contact facet that is an indexer property.
Index name for indexer property	Name of the key in the dictionary used to store the specific value.
Property name	Name of the property on the object read from the dictionary whose value is accessed.

9.2.3 xDB Contact Facet Collection Property Value Accessor

This Value Accessor is used for properties that are collections.

Field	Description
Contact facet name	Name of the contact facet to access.
Property name	Name of the property on the contact facet.
Value appender type	Type that is able to add a value to the collection. In most cases you will not need to change this value.
Do not use value appender	Specifies whether or not the Value Appender is used. Enable this value if you want to overwrite the value on the contact facet. An example is when you want to reset the value. Disable this value if you want to use the Value Appender. This will allow you to add values to the collection without losing values that already exist in the collection.

9.2.4 xDB Identifier Value Accessor

This Value Accessor is used to access the contact identifier. There are no fields on this template because there is nothing to configure.