Anjep Consulting ApS

CRM GDPR Audit Manual  
version 0.1

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Author | Description |
| 2018-06-29 | 0.1 | Łukasz Grzybowski-Glikman | First draft |
| 2018-08-16 | 0.2 | Łukasz Grzybowski-Glikman | Added a simple sample setup scenario. |
|  |  |  |  |

Contents

[2 Features 5](#_Toc526248706)

[3 How does it work? 6](#_Toc526248707)

[4 Installation 7](#_Toc526248708)

[5 Configuration 8](#_Toc526248709)

[5.1 Audit Configuration 8](#_Toc526248710)

[5.2 Entity Filters 9](#_Toc526248711)

[5.3 Audit Field Filter 10](#_Toc526248712)

[5.4 Audit Target 11](#_Toc526248713)

[5.5 Blacklist Users 12](#_Toc526248714)

[6 Audit Targets 13](#_Toc526248715)

[6.1 Local CRM 13](#_Toc526248716)

[6.2 Web Proxy 14](#_Toc526248717)

[6.2.1 Basic information 14](#_Toc526248718)

[6.2.2 Request model 14](#_Toc526248719)

[6.2.3 Api Key 16](#_Toc526248720)

[6.2.4 Example CURL request 17](#_Toc526248721)

[6.2.5 Data passed to the service 18](#_Toc526248722)

[6.2.6 Example of request 19](#_Toc526248723)

[6.2.7 Using the out of the box Proxy Web Service for logging in SQL 21](#_Toc526248724)

[6.3 Azure Tables 23](#_Toc526248725)

[6.3.1 Basic information 23](#_Toc526248726)

[6.3.2 Configuration 23](#_Toc526248727)

[6.4 SQL 26](#_Toc526248728)

[6.4.1 Basic information 26](#_Toc526248729)

[6.4.2 Setup the database 26](#_Toc526248730)

[6.4.3 Changing isolation mode to not isolated 27](#_Toc526248731)

[6.4.4 Configuration 29](#_Toc526248732)

[7 Format of data logged in SQL 31](#_Toc526248733)

[7.1 Query XML – Format 31](#_Toc526248734)

[7.2 Query XML – Example for Retrieve 32](#_Toc526248735)

[7.3 Query XML – Example for RetrieveMultiple 33](#_Toc526248736)

[7.4 Data XML - Format 34](#_Toc526248737)

[7.5 Data XML – Example 35](#_Toc526248738)

[7.6 Data XML – Querying 36](#_Toc526248739)

[8 Example setup 37](#_Toc526248740)

[8.1 Installation 37](#_Toc526248741)

[8.2 Create a basic configuration 37](#_Toc526248742)

[8.3 Testing 40](#_Toc526248743)

[9 Error handling 41](#_Toc526248744)

# Features

The Anjep Consulting ApS CRM GDPR Audit solution makes it possible to log all user access to CRM data.

Features include:

1. Filtering of entities and fields included in audit log.
2. Several targets for storing the audit log data, including CRM, SQL, Azure Tables and a generic Web Service proxy.
3. Excluding certain users from being audited.
4. Minimal impact on the system performance due to code optimization and asynchronous operations. From a system user perspective enabling the solution shouldn’t have any visible performance impact.

# How does it work?

The CRM GDPR audit registers plugins on all CRM data retrieval operations. When data is retrieved it compares it with the configured filters. If the result is that it should be logged it is processed and send to the audit target (SQL, CRM, etc.) for storage. An important note is that the plugins have no dependencies on data stored in CRM. All processing is done “in memory”, thus avoiding any queries to CRM. This allows it to have a minimal impact on the systems performance.

Additionally, the plugins work asynchronously. This means that even in the unlikely event of failure (maybe due to a log target unavailability) it will not cause any disruptions in the operation of the system. The errors will only be visible for administrators when looking at the “System Jobs” log.

It is possible to configure several audit targets. So if for example the database is unavailable, the operation will still be logged to CRM.

# Installation

The CRM GDPR audit solution can either be installed through the Microsoft AppSource add-on store or manually through importing a managed solution.

Please make sure you **Publish All Customization** after installing the solution. We have observed that failure to do so can result in errors when trying to run it.

The Publish All Customizations options is available inside Dynamics 365\Settings\Customizations\Solutions.

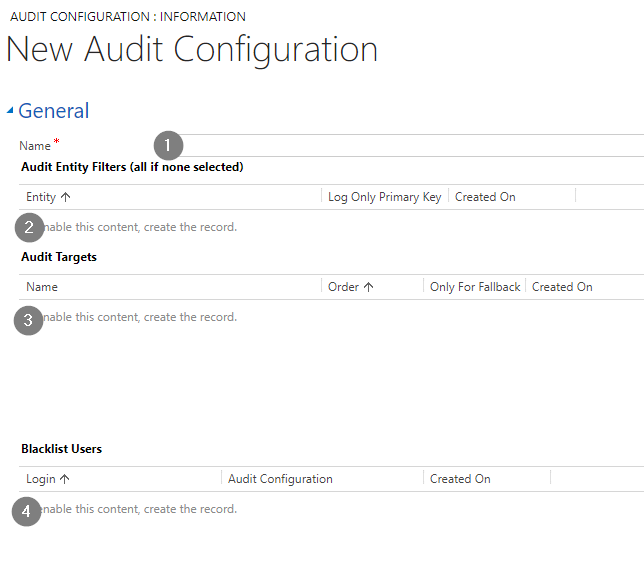


# Configuration

## Audit Configuration

The auditing is configured by creating new records of the Audit Configuration entity.

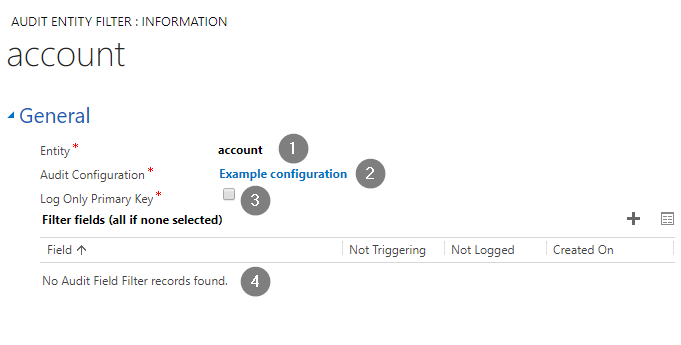
There can be multiple active configurations. All will be processed independently.



|  |  |  |
| --- | --- | --- |
| No. | Element | Description |
| 1 | Name | The name for the configuration. Only for information purposes. |
| 2 | Audit Entity Filters | Filters the list of entities that will be audited (access to which will be logged). If none are selected the solution will log access to all entities in the system. |
| 3 | Audit Targets | Determine where the audit information will be persisted. Each configuration can have multiple targets (thus storing the log in several places). A target can be primary of fallback. Fallback targets are only uses in case the primary fail.  Until at least one target is added, nothing will be logged. |
| 4 | Blacklist users | A list of users which will be excluded from the auditing. This could be integration users, administrators, SYSTEM, etc. By default this list is empty. |

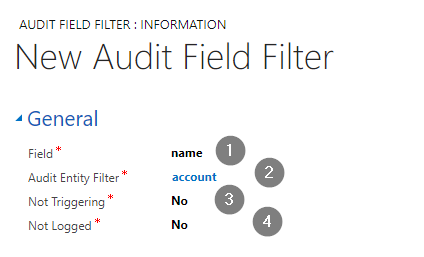
## Entity Filters

If at least one entity is set in the filter, only access to those that are added will be logged.



|  |  |  |
| --- | --- | --- |
| No. | Element | Description |
| 1 | Entity | The schema name of the entity that should be included in the filter. |
| 2 | Audit Configuration | Lookup to the configuration. Under normal circumstances there is no need to change this. |
| 3 | Log Only Primary Key | If set only the primary key of the entity (GUID identifier) will be included in the log. This is the easiest way to ensure the log doesn’t itself include any sensitive information. |
| 4 | Filter Fields | A list of fields access to which will be audited or which will trigger the log. If none are selected access to any field will be logged. If the “Log Only Primary Key” option is not checked this will also include the values of those fields. |

## Audit Field Filter



|  |  |  |
| --- | --- | --- |
| No. | Element | Description |
| 1 | Filed | The schema name of the field that should be included in the filter. |
| 2 | Audit Entity Filter | Lookup to the Entity Filter. Under normal circumstances there is no need to change this. |
| 3 | Not Triggering | If only fields marked as “Not Triggering” are retrieved it will not be logged. Although if in the same operation any other field that is not marked as “Not Triggering” is retrieved, they will be included in the log. |
| 4 | Not Logged | The value of the field will not be logged, although it will trigger the creation of a log entry. |

Possible combinations:

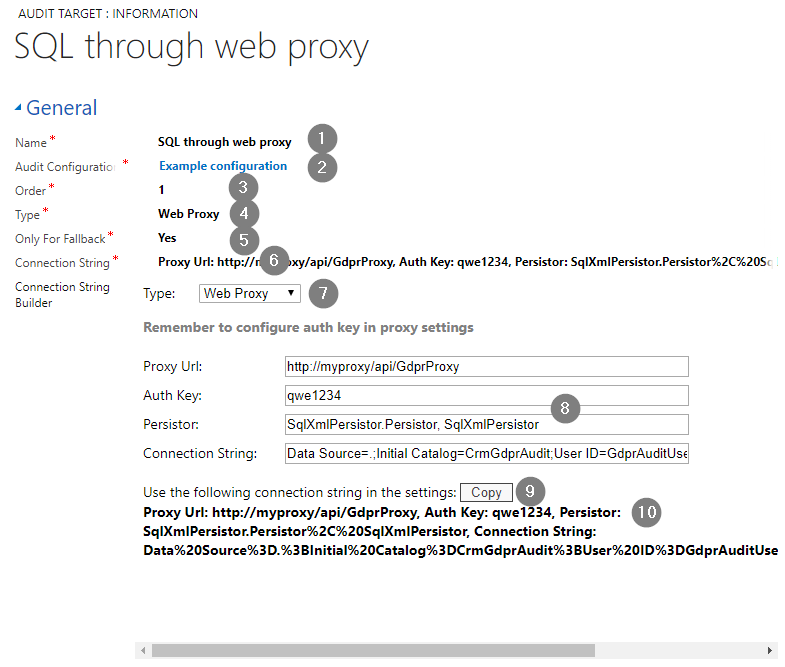
|  |  |  |
| --- | --- | --- |
| Not Triggering | Not Logged | Result |
| No | **No** | Triggers log and value is logged. |
| No | **Yes** | Triggers log, but value is not logged. |
| Yes | **No** | Doesn’t trigger log, but value is logged if triggered by another field. |
| Yes | **Yes** | Doesn’t trigger log and value is not logged. Same as if the field was not added at all. |

## Audit Target

Currently the solution supports the following targets for storing audit logs:

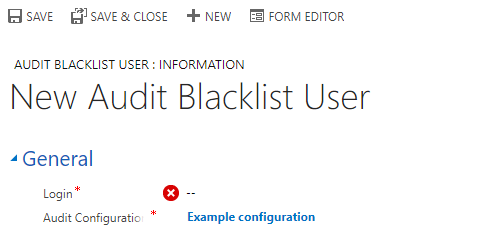
|  |  |
| --- | --- |
| Name | Description |
| Local CRM | Data is stored inside CRM, in the entity Audit Entry. |
| Web Proxy | Data to be stored is sent to a webservice. The solution includes a sample implementation of such a web service, which is supposed to be used primarily for SQL logging. Although this can be any web service. The only limitations are it needs to implement the required interface, be accessible from the CRM server and hosted with https enabled. |
| Azure Tables | Data is stored in Azure Table storage. |
| SQL | Data is sent directly to SQL. This requires the plugins to run in non-isolated mode and is only with on-premise deployments. |

Details on how to configure each target are provided in a dedicated section of this document.



|  |  |  |
| --- | --- | --- |
| No. | Element | Description |
| 1 | Name | The name for the target. Only for information purposes. |
| 2 | Audit Configuration | Lookup to the configuration. Under normal circumstances there is no need to change this. |
| 3 | Order | The order in which this target should be executed. If several targets have the same order number they will be executed in random order. |
| 4 | Type | The type of the target: Local CRM, Web Proxy, Azure Tables, SQL, … |
| 5 | Only For Fallback | Indicates that this target should only be used if there has been an error with the previous targets (not marked as fallback). This allows for a setup where, for example, the primary target is SQL, but if it fails data will be logged in CRM. |
| 6 | Connection String | The connection string for the specific type of target. |
| 7 | Connection String Builder | Helper tool for building the connection string. |
| 8 | Connection String Builder options | Each target has a different kind of connection string. The helper tools shows option specific for that target type and builds the connection string as you type. |
| 9 | Copy | Copy the dynamically built connection string to the clipboard. |
| 10 | Built Connection String | Dynamically build connection string. Use this in the main “Connection String” field of the configuration. |

## Blacklist Users

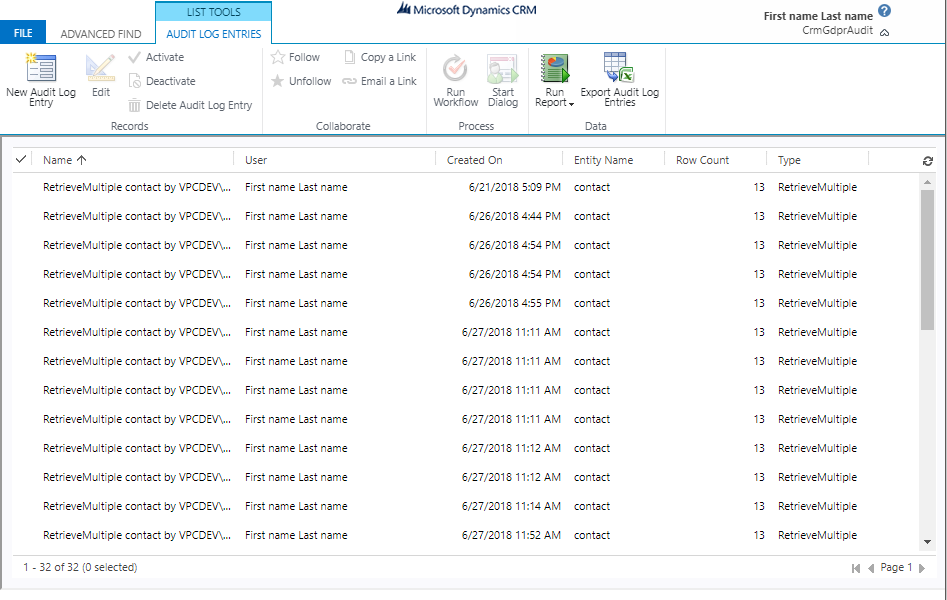


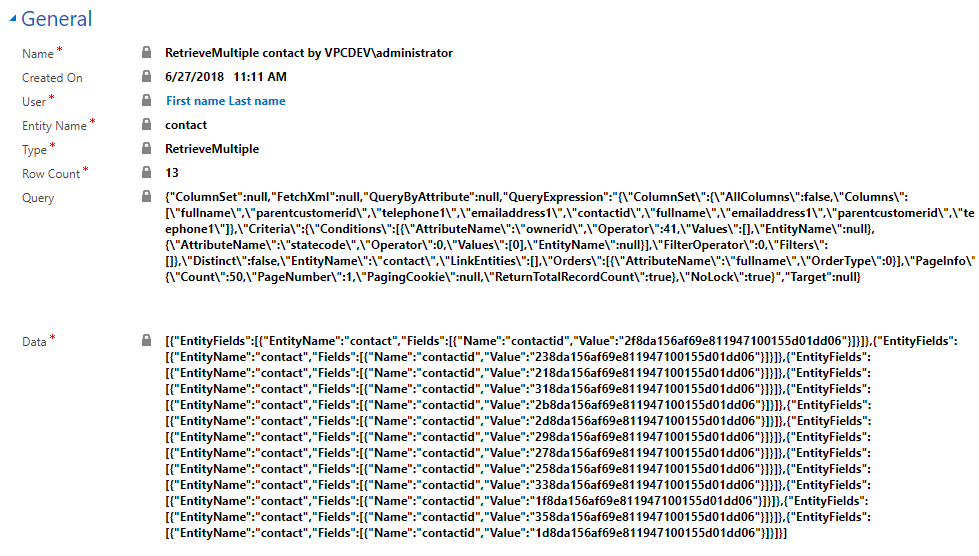
Provide the login name of user that should be black listed. Each record is one user.

# Audit Targets

## Local CRM

The basic target, which stores audit information directly in CRM. Not recommended because of performance impact on the system (consider there can be thousands or more log entries every minute), but a good fallback because “it should always work” and has no dependencies on external components.





## Web Proxy

### Basic information

The Web Proxy target calls a web service it is pointed to whenever an audit log needs to be persisted. There are two main purposes for that:

1. Allowing (build in) SQL logging, when the solution plugins are registered in isolated mode
2. Allowing the user of the solution to provide his own log persistence target.

The Web Proxy service needs to be a REST based web service, which the CRM server can contact and configured with an https binding (limitation of CRM isolated / sandboxed plugins).

Part of the solution is a sample implementation of such a service. Which can be used to log data to SQL.

### Request model

The web service is called using HTTP POST with data inside the body, with the following schema (generated through Swagger):

PersistRequest {

PersistorQualifiedName (string, optional),

ConnectionString (string, optional),

DataRetrievalEvent (DataRetrievalEvent, optional)

}

DataRetrievalEvent {

TimeOfRetrievalUtc (string, optional),

DataRetrievedBy (User, optional),

BaseQueryEntityName (string, optional),

Query (Query, optional),

Type (string, optional),

RowCount (integer, optional),

Records (Array[Record], optional)

}

User {

Id (string, optional),

Login (string, optional),

FullName (string, optional)

}

Query {

FetchXml (string, optional),

QueryExpression (string, optional),

QueryByAttribute (string, optional),

Target (string, optional),

ColumnSet (string, optional)

}

Record {

EntityFields (Array[EntityFields], optional)

}

EntityFields {

EntityName (string, optional),

Fields (Array[Field], optional)

}

Field {

Name (string, optional),

Value (string, optional)

}

**Example:**

{

"PersistorQualifiedName": "string",

"ConnectionString": "string",

"DataRetrievalEvent": {

"TimeOfRetrievalUtc": "2018-06-29T10:07:53.876Z",

"DataRetrievedBy": {

"Id": "00000000-0000-0000-0000-000000000000",

"Login": "string",

"FullName": "string"

},

"BaseQueryEntityName": "string",

"Query": {

"FetchXml": "string",

"QueryExpression": "string",

"QueryByAttribute": "string",

"Target": "string",

"ColumnSet": "string"

},

"Type": "string",

"RowCount": 0,

"Records": [

{

"EntityFields": [

{

"EntityName": "string",

"Fields": [

{

"Name": "string",

"Value": "string"

}

]

}

]

}

]

}

}

### Api Key

When the web service is called a header value named **apiKey** will be added. This can be used to secure the web service and not make it publicly available.

More complex authentication schemes might be added in future versions of this solution.

### Example CURL request

curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --header 'apiKey: test api key' -d '{ \

"PersistorQualifiedName": "string", \

"ConnectionString": "string", \

"DataRetrievalEvent": { \

"TimeOfRetrievalUtc": "2018-06-29T10:07:53.876Z", \

"DataRetrievedBy": { \

"Id": "00000000-0000-0000-0000-000000000000", \

"Login": "string", \

"FullName": "string" \

}, \

"BaseQueryEntityName": "string", \

"Query": { \

"FetchXml": "string", \

"QueryExpression": "string", \

"QueryByAttribute": "string", \

"Target": "string", \

"ColumnSet": "string" \

}, \

"Type": "string", \

"RowCount": 0, \

"Records": [ \

{ \

"EntityFields": [ \

{ \

"EntityName": "string", \

"Fields": [ \

{ \

"Name": "string", \

"Value": "string" \

} \

] \

} \

] \

} \

] \

} \

}' 'http://localhost:54769/api/GdprAuditProxy'

### Data passed to the service

|  |  |  |
| --- | --- | --- |
| No. | Element | Description |
| 1 | PersistorQualifiedName | Notifies the service which persistor the user wants to use. Could be values like “SQL”, “Text”, etc. |
| 2 | ConnectionString | Connection string for the chosen persistor. |
| 3 | DataRetrievalEvent\TimeOfRetrievalUtc | UTC date and time of when the data was retrieved. |
| 4 | DataRetrievalEvent\DataRetrievedBy\Id | CRM ID of the user retrieving the data. |
| 5 | DataRetrievalEvent\DataRetrievedBy\Login | Login of the user retrieving the data. |
| 6 | DataRetrievalEvent\DataRetrievedBy\FullName | Full name of the user retrieving the data. |
| 7 | DataRetrievalEvent\BaseQueryEntityName | Schema name of the base entity of the query. |
| 8 | DataRetrievalEvent\Query\FetchXml | Fetch XML (if present) used for the query. JSON serialized. |
| 9 | DataRetrievalEvent\Query\QueryExpression | Query expression (if present) used for the query. JSON serialized. |
| 10 | DataRetrievalEvent\Query\QueryByAttribute | Query by attribute (if present) used for the query. JSON serialized. |
| 11 | DataRetrievalEvent\Query\Target | Target entity of the query. |
| 12 | DataRetrievalEvent\Query\ColumnSet | Column set used in the query. JSON serialized. |
| 13 | DataRetrievalEvent\Type | Type of event (Retrieve, RetrieveMultiple, QueryByAttribute, Fetch) |
| 14 | DataRetrievalEvent\RowCount | Number of rows retrieved. |
| 15 | DataRetrievalEvent\Records[]\EntityFields[]\EntityName | Result entity schema name. |
| 16 | DataRetrievalEvent\Records[]\EntityFields[]\Fields[]\Name | Name of the field. |
| 17 | DataRetrievalEvent\Records[]\EntityFields[]\Fields[]\Value | Retrieved value. |

In the example implementation the web service is truly a “proxy”. That’s why it’s passed both the assembly name of the persistor to use and the connection string for that persistor. It doesn’t implement any logic of it’s own except basic Web API functionality. It’s not aware of what it’s logging and where. The implementation can off course vary.

### Example of request

Below is a full example request. For a RetrieveMultiple type query, for Contacts with joined Accounts.

{

"PersistorQualifiedName": "SqlXmlPersistor.Persistor, SqlXmlPersistor",

"ConnectionString": "Data Source=.;Initial Catalog=CrmGdprAudit;User ID=GdprLogUser;Password=MyPassword;",

"DataRetrievalEvent": {

"TimeOfRetrievalUtc": "2018-06-29T11:11:12.751Z",

"DataRetrievedBy": {

"Id": "d7137f12-ae69-e811-9471-00155d01dd06",

"Login": "VPCDEV\\administrator",

"FullName": "First name Last name"

},

"BaseQueryEntityName": "contact",

"Query": {

"FetchXml": null,

"QueryExpression": "{\"ColumnSet\":{\"AllColumns\":false,\"Columns\":[\"fullname\",\"parentcustomerid\",\"telephone1\",\"emailaddress1\",\"contactid\",\"fullname\",\"emailaddress1\",\"parentcustomerid\",\"telephone1\"]},\"Criteria\":{\"Conditions\":[{\"AttributeName\":\"ownerid\",\"Operator\":41,\"Values\":[],\"EntityName\":null},{\"AttributeName\":\"statecode\",\"Operator\":0,\"Values\":[0],\"EntityName\":null}],\"FilterOperator\":0,\"Filters\":[]},\"Distinct\":false,\"EntityName\":\"contact\",\"LinkEntities\":[{\"Columns\":{\"AllColumns\":false,\"Columns\":[\"address1\_city\",\"accountnumber\",\"name\"]},\"EntityAlias\":\"a\_dc9b80f8c78146d89fd6a3b610836975\",\"JoinOperator\":1,\"LinkCriteria\":{\"Conditions\":[],\"FilterOperator\":0,\"Filters\":[]},\"LinkEntities\":[],\"LinkFromAttributeName\":\"parentcustomerid\",\"LinkFromEntityName\":\"contact\",\"LinkToAttributeName\":\"accountid\",\"LinkToEntityName\":\"account\"}],\"Orders\":[{\"AttributeName\":\"fullname\",\"OrderType\":0}],\"PageInfo\":{\"Count\":50,\"PageNumber\":1,\"PagingCookie\":null,\"ReturnTotalRecordCount\":true},\"NoLock\":true}",

"QueryByAttribute": null,

"Target": null,

"ColumnSet": null

},

"Type": "RetrieveMultiple",

"RowCount": 2,

"Records": [

{

"EntityFields": [

{

"EntityName": "contact",

"Fields": [

{

"Name": "emailaddress1",

"Value": "someone\_j@example.com"

},

{

"Name": "parentcustomerid",

"Value": "c98ca156af69e811947100155d01dd06"

},

{

"Name": "telephone1",

"Value": "555-0109"

},

{

"Name": "fullname",

"Value": "Jim Glynn (sample)"

},

{

"Name": "contactid",

"Value": "2f8da156af69e811947100155d01dd06"

}

]

},

{

"EntityName": "account",

"Fields": [

{

"Name": "accountnumber",

"Value": "BABCO88H"

},

{

"Name": "name",

"Value": "Coho Winery (sample)"

},

{

"Name": "address1\_city",

"Value": "Phoenix"

}

]

}

]

},

{

"EntityFields": [

{

"EntityName": "contact",

"Fields": [

{

"Name": "emailaddress1",

"Value": "someone\_c@example.com"

},

{

"Name": "parentcustomerid",

"Value": "bb8ca156af69e811947100155d01dd06"

},

{

"Name": "telephone1",

"Value": "555-0102"

},

{

"Name": "fullname",

"Value": "Nancy Anderson (sample)"

},

{

"Name": "contactid",

"Value": "218da156af69e811947100155d01dd06"

}

]

},

{

"EntityName": "account",

"Fields": [

{

"Name": "accountnumber",

"Value": "ABC28UU7"

},

{

"Name": "name",

"Value": "Adventure Works (sample)"

},

{

"Name": "address1\_city",

"Value": "Santa Cruz"

}

]

}

]

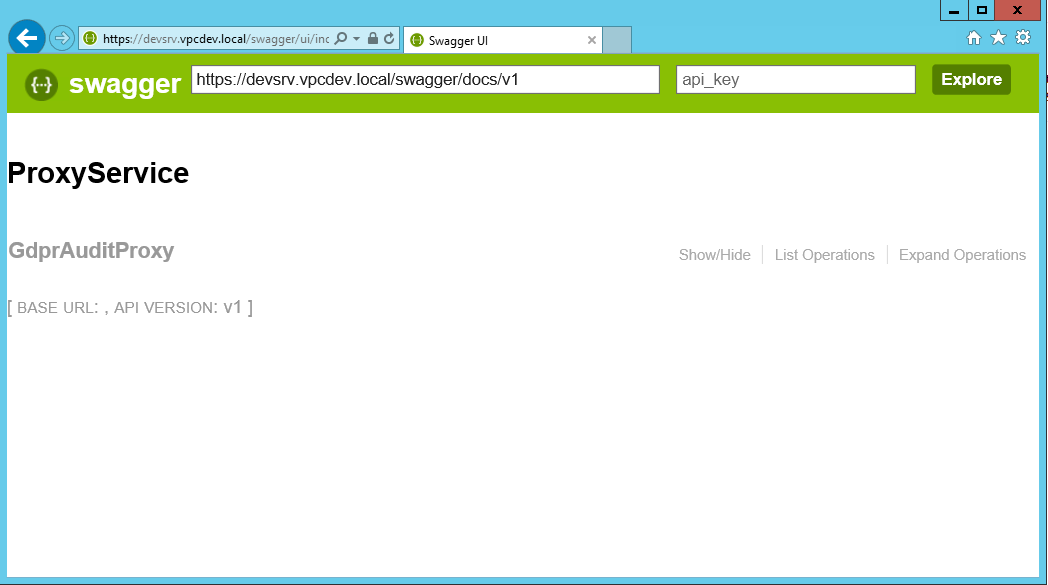
}

]

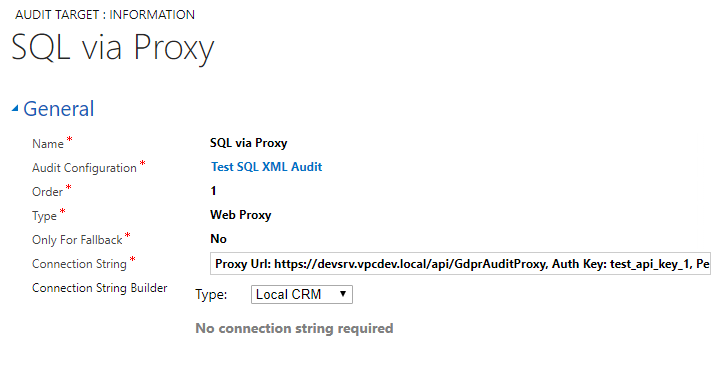
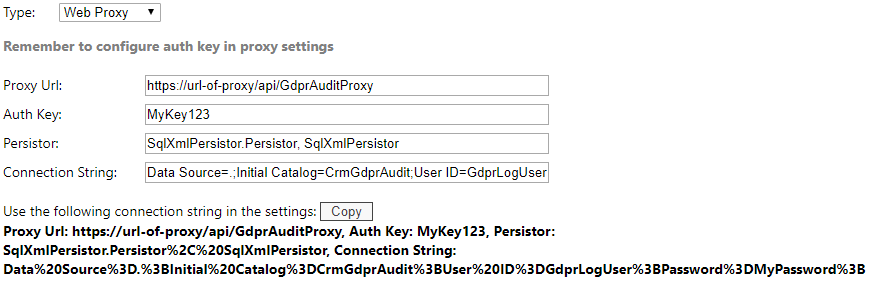
}

}

### Using the out of the box Proxy Web Service for logging in SQL

1. Setup the database as described in the section of this document regarding the SQL target.
2. Deploy the web service anywhere the CRM server can talk to. For on-premise deployments the easiest is to do that directly on the CRM front end server. This avoids any issues with firewalls etc.
3. Make sure it’s using an https binding. For on-premise deployments this can be a self-signed certificate.
4. Configure the authentication keys inside the file AppData\ApiKeys.json
5. Test if everything is setup correctly by opening the proxy via a browser, using http://url-to-proxy. It should show a page like this:  
   

There shouldn’t be any errors related to the certificate. Else it will cause the logging to fail. Testing in Internet Explorer is recommended.

1. Configure the target like this:  
     
     
   For the connection string the following values should be used:  
   

|  |  |  |
| --- | --- | --- |
| No. | Element | Value |
| 1 | Proxy URL | https://url-of-proxy/api/GdprAuditProxy |
| 2 | Auth Key | One of the configured authentication keys |
| 3 | Persistor | UTC date and time of when the data was retrieved. |
| 4 | Connection String | Regular MS SQL connection string. See <https://www.connectionstrings.com/sql-server/> for examples. SQL User based authentication is recommended. |

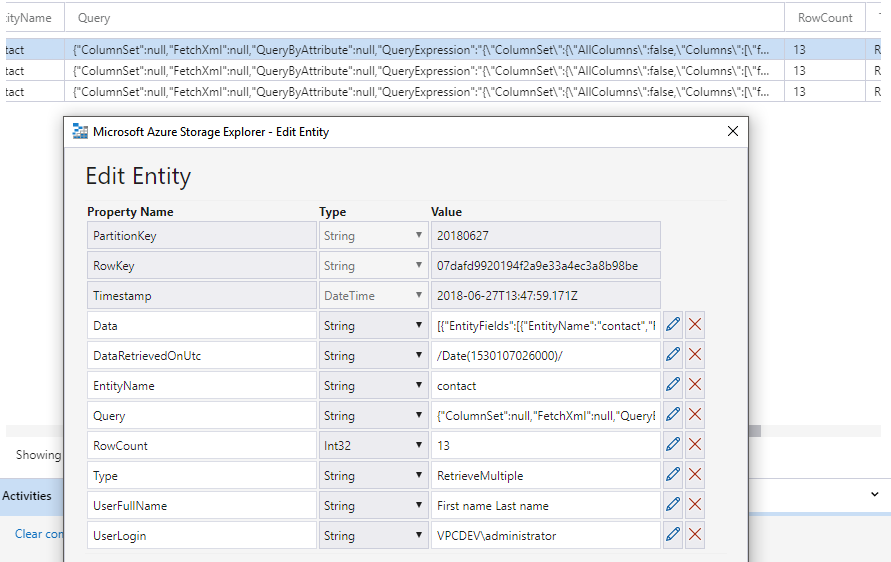
1. Test the operation in CRM and see if data is logged. See the section describing error handling for more details on handling errors. Optionally the web proxy can be tested in isolation through it’s Swagger UI or tools like Postman. The example request shown above can be used for testing.

## Azure Tables

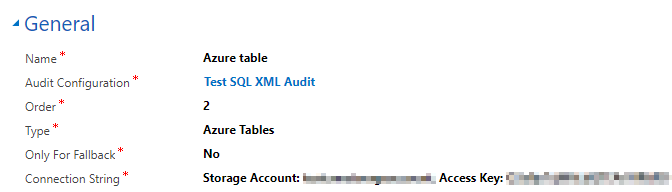
### Basic information

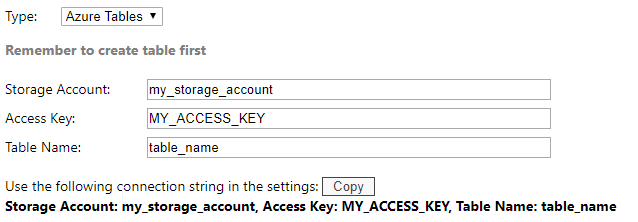
The Azure Tables target is a build in way of storing data in Azure Tables. Because of the distributed cloud architecture high throughput and scalability are provided out of the box. Options like redundancy and geo-scaling are also optionally available. The format the data is stored is identical to the SQL Server one. Basic information is stored in columns, but the query and data (which have a dynamic schema) are serialized.

Below is an example on how such data might look (obtained using the Microsoft Azure Storage Explorer tool):



### Configuration



For the connection string the following values should be used:  


These values can be obtained from the Azure Portal.

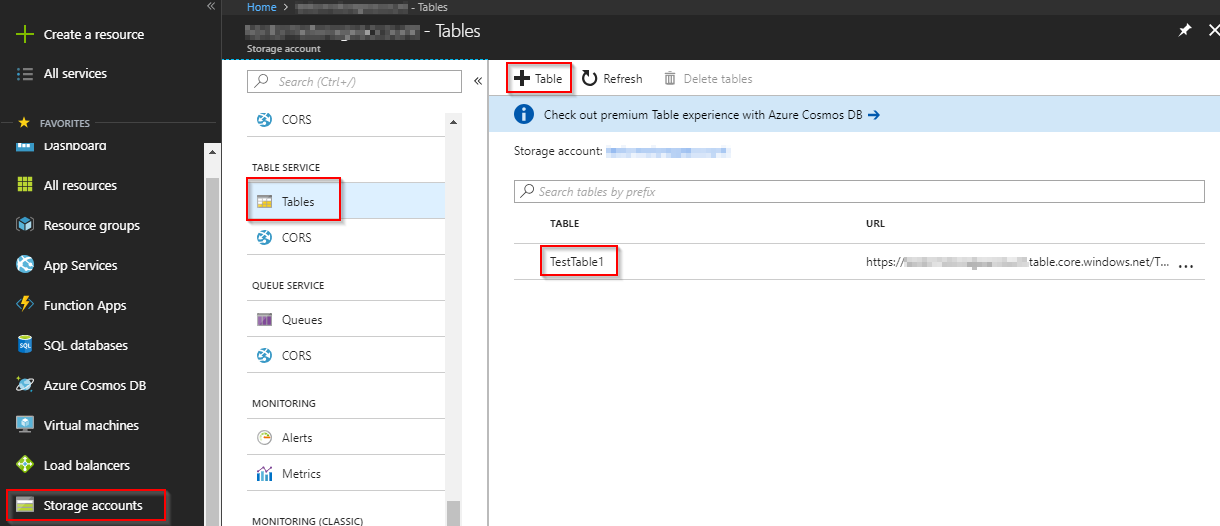


Figure 1 Create a table. Copy the table name.

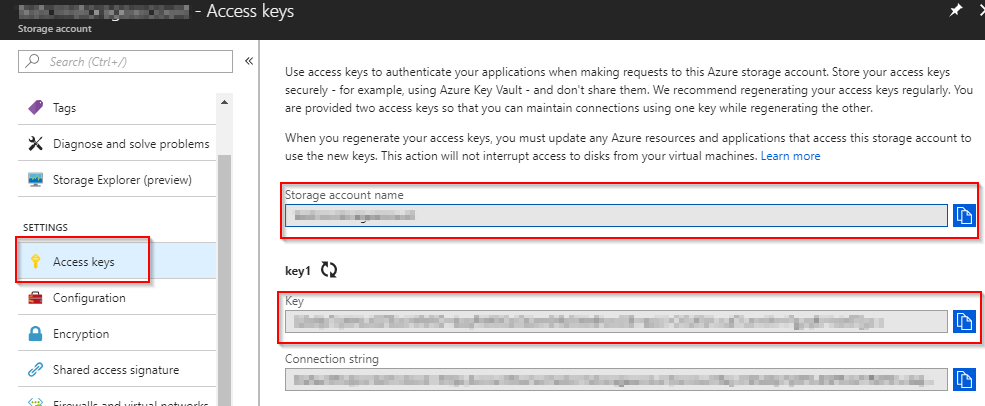


Figure 2 Copy the storage account name and access key

## SQL

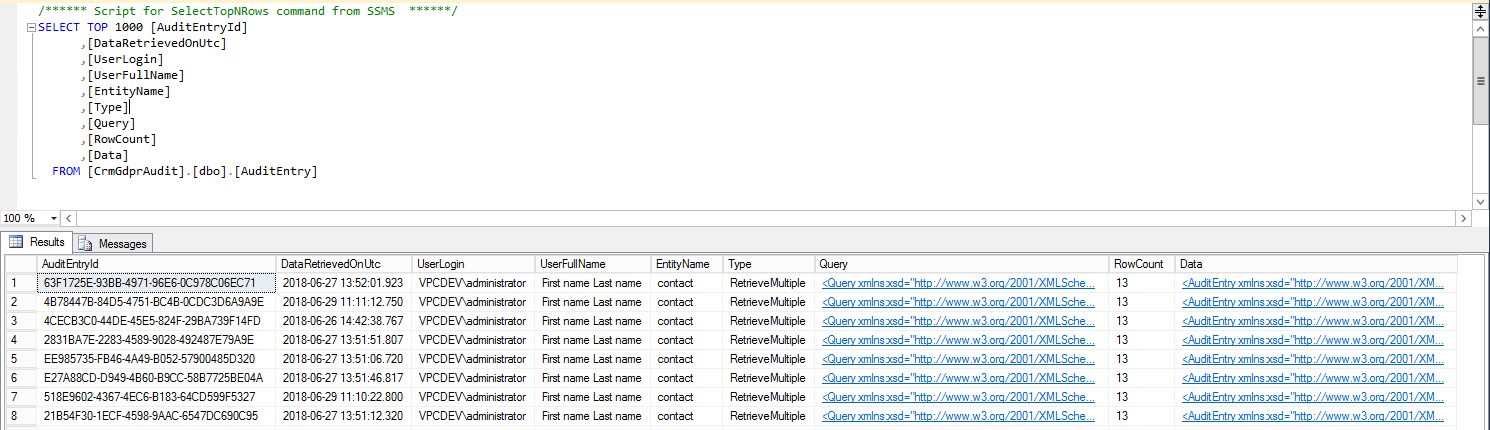
### Basic information

Allows for storing data directly in SQL server.

This will only work in on-premise versions of CRM after the GDPR Audit Logging have been configured to run with no isolation.

This can be done through the plugin registration tool or through a dialog on the configuration. Although the dialog is on a single configuration it affects global registration of the plugins.

The result should look close to the one below:



Note that the data is serialized and stored as XML. This means it can be queried directly in SQL using it’s built in XML query capabilities - <https://docs.microsoft.com/en-us/sql/xquery/xquery-language-reference-sql-server>

### Setup the database

The build in SQL target requires just a single table, created using this script. For connection it’s recommended to create a SQL user with just write permissions to this single table.

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[AuditEntry](

[AuditEntryId] [uniqueidentifier] NOT NULL,

[DataRetrievedOnUtc] [datetime] NOT NULL,

[UserLogin] [nvarchar](100) NOT NULL,

[UserFullName] [nvarchar](100) NOT NULL,

[EntityName] [nvarchar](100) NOT NULL,

[Type] [nvarchar](50) NOT NULL,

[Query] [xml] NOT NULL,

[RowCount] [int] NOT NULL,

[Data] [xml] NOT NULL,

CONSTRAINT [PK\_AuditEntry] PRIMARY KEY CLUSTERED

(

[AuditEntryId] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [PRIMARY]

) ON [PRIMARY] TEXTIMAGE\_ON [PRIMARY]

GO

ALTER TABLE [dbo].[AuditEntry] ADD CONSTRAINT [DF\_AuditEntry\_AuditEntryId] DEFAULT (newid()) FOR [AuditEntryId]

GO

Depending on the types of queries you’ll be performing against this table it might be reasonable to add some non-clustered indexes. None are added by default, because they should depend on the actual query criteria. The DataRetrievedOnUtc and UserLogin columns are a suggestion.

### Changing isolation mode to not isolated

Run the helper dialog:



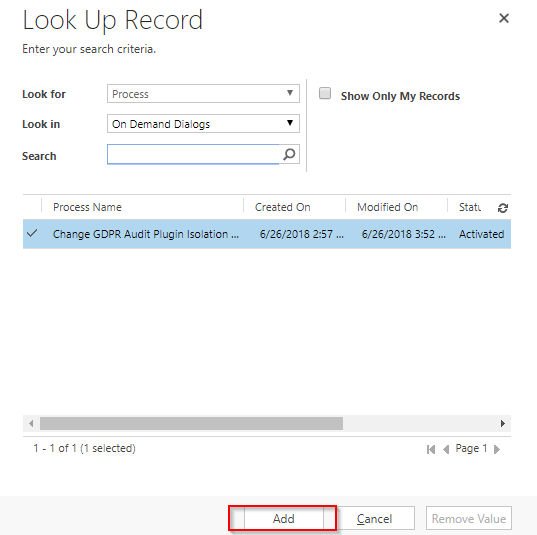


Figure 3 Run dialog

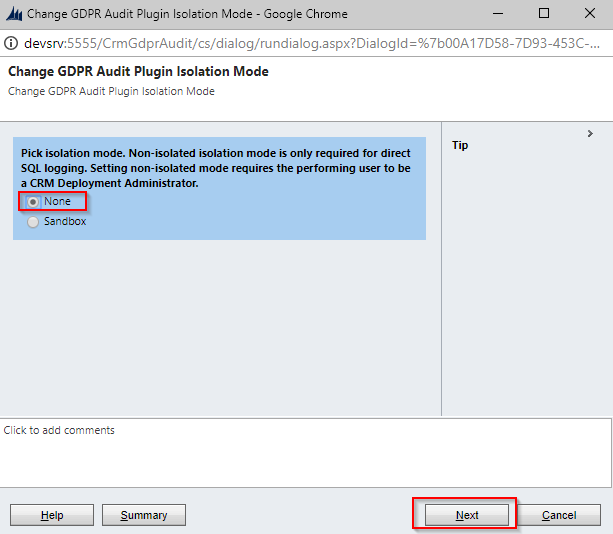
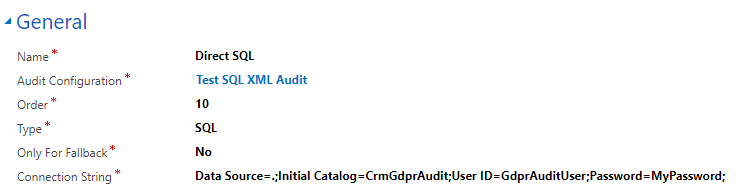


Figure 4 Choose "None" and click [Next]

Please note that the user running the dialog must have Deployment Adminsitrator permissions, to be able to change the plugins isolation mode to None.

### Configuration

An example configuration should look something like this:



The connection string is a regular MS SQL connection string. See <https://www.connectionstrings.com/sql-server/> for examples. SQL User based authentication is recommended.

# Format of data logged in SQL

This section describes the way data is stored when using SQL logging (either direct or through a web service proxy).

|  |  |  |
| --- | --- | --- |
| Column | Data type | Description |
| AuditEntryId | UniqueIdentifier (Guid) | Primary key of table |
| DataRetrievedOnUtc | DateTime | Date and time on which data was retrieved in UTC |
| UserLogic | NVarChar(100) | Login of user who retrieved the data |
| UserFullName | NVarChar(100) | Full name of user who retrieved the data |
| EntityName | NVarChar(100) | The name of the top level entity from which the data was retrieved |
| Type | NVarChar(50) | Type of query – Retrieve, RetrieveMultiple or Fetch |
| Query | XML | Serialized query. Content depends on type. Described later. |
| RowCount | Int | Nr of retrieved rows |
| Data | XML | Retrieved data after filtering as defined in the configuration. Described later. |

## Query XML – Format

The format of the stored data is the following:

* Query
  + FetchXml
  + QueryExpression
  + QueryByAttribute
  + Target
  + ColumnSet

Which fields will be filled depends on the type of query.

|  |  |
| --- | --- |
| Query type | Filled fields |
| Retrieve | Target (Entity), ColumnSet |
| RetrieveMultiple | QueryExpression or QueryByAttribute or FetchExpression (depends on the original query) |
| Fetch | FetchXml |

The inner value is an JSON serialization of the query type. Exact attributes can be looked up in the CRM SDK.

## Query XML – Example for Retrieve

<Query xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<Target>

{

"Id":"278da156-af69-e811-9471-00155d01dd06",

"LogicalName":"contact",

"Name":null

}

</Target>

<ColumnSet>

{

"AllColumns":false,

"Columns":[

"ownerid",

"fullname",

"jobtitle",

"parentcustomerid",

"emailaddress1",

"telephone1",

"mobilephone",

"fax",

"preferredcontactmethodcode",

"address1\_composite",

"gendercode",

"familystatuscode",

"spousesname",

"birthdate",

"anniversary",

"description",

"originatingleadid",

"lastusedincampaign",

"donotsendmm",

"donotemail",

"donotbulkemail",

"donotphone",

"donotfax",

"donotpostalmail",

"transactioncurrencyid",

"creditlimit",

"creditonhold",

"paymenttermscode",

"address1\_shippingmethodcode",

"address1\_freighttermscode",

"address1\_addressid",

"address2\_addressid",

"statecode",

"address1\_line1",

"address1\_line2",

"address1\_line3",

"address1\_city",

"address1\_stateorprovince",

"address1\_postalcode",

"address1\_country",

"firstname",

"middlename",

"lastname",

"entityimage\_url",

"processid",

"stageid",

"contactid",

"fullname",

"ownerid",

"statecode",

"statuscode"

]

}

</ColumnSet>

</Query>

## Query XML – Example for RetrieveMultiple

<Query xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<QueryExpression>

{

"ColumnSet":{

"AllColumns":false,

"Columns":[

"fullname",

"parentcustomerid",

"telephone1",

"emailaddress1",

"contactid",

"fullname",

"emailaddress1",

"parentcustomerid",

"telephone1"

]

},

"Criteria":{

"Conditions":[

{

"AttributeName":"ownerid",

"Operator":41,

"Values":[

],

"EntityName":null

},

{

"AttributeName":"statecode",

"Operator":0,

"Values":[

0

],

"EntityName":null

}

],

"FilterOperator":0,

"Filters":[

]

},

"Distinct":false,

"EntityName":"contact",

"LinkEntities":[

],

"Orders":[

{

"AttributeName":"fullname",

"OrderType":0

}

],

"PageInfo":{

"Count":250,

"PageNumber":1,

"PagingCookie":null,

"ReturnTotalRecordCount":true

},

"NoLock":true

}

</QueryExpression>

</Query>

## Data XML - Format

The format of the stored data is the following:

* AuditEntry (top level)
  + Records (groups records)
    - Record
      * EntityFields (grouping)
        + EntityFields (one per entity per record)

EntityName (text – entity logical name)

Fields (grouping)

Field

Name

Value

Each record can contain multiple entities same way as a single result from CRM can. This is because of LinkedEntities (SQL JOINs).

## Data XML – Example

<AuditEntry xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<Records>

<Record>

<EntityFields>

<EntityFields>

<EntityName>contact</EntityName>

<Fields>

<Field>

<Name>fullname</Name>

<Value>Jim Glynn (sample)</Value>

</Field>

<Field>

<Name>parentcustomerid</Name>

<Value>c98ca156af69e811947100155d01dd06</Value>

</Field>

<Field>

<Name>contactid</Name>

<Value>2f8da156af69e811947100155d01dd06</Value>

</Field>

</Fields>

</EntityFields>

<EntityFields>

<EntityName>account</EntityName>

<Fields>

<Field>

<Name>name</Name>

<Value>Test parent account</Value>

</Field>

<Field>

<Name>accountid</Name>

<Value>c98ca156af69e811947100155d01dd06</Value>

</Field>

</Fields>

</EntityFields>

</EntityFields>

</Record>

<Record>

<EntityFields>

<EntityFields>

<EntityName>contact</EntityName>

<Fields>

<Field>

<Name>fullname</Name>

<Value>Maria Campbell (sample)</Value>

</Field>

<Field>

<Name>parentcustomerid</Name>

<Value>bd8ca156af69e811947100155d01dd06</Value>

</Field>

<Field>

<Name>telephone1</Name>

<Value>555-0103</Value>

</Field>

</Fields>

</EntityFields>

</EntityFields>

</Record>

</Records>

</AuditEntry>

## Data XML – Querying

The Data XML can be queried using regular T-SQL methods for dealing with XML. For example, the following query will return a list of contacts that have been accessed by a given user at a given date.

SELECT

Data.query('/AuditEntry/Records/Record/EntityFields/EntityFields/Fields/Field[Name="contactid"]/Value') AS ContactIds

FROM AuditEntry

WHERE

UserLogin = 'VPCDEV\administrator'

AND DataRetrievedOnUtc >= '2018-09-26' AND DataRetrievedOnUtc < '2018-09-30'

# Example setup

This section will describe how to create a simple sample setup of the audit logging to verify the solution operation correctly.

The process will be described step by step, starting with a clean instance of Dynamics 365 Online.

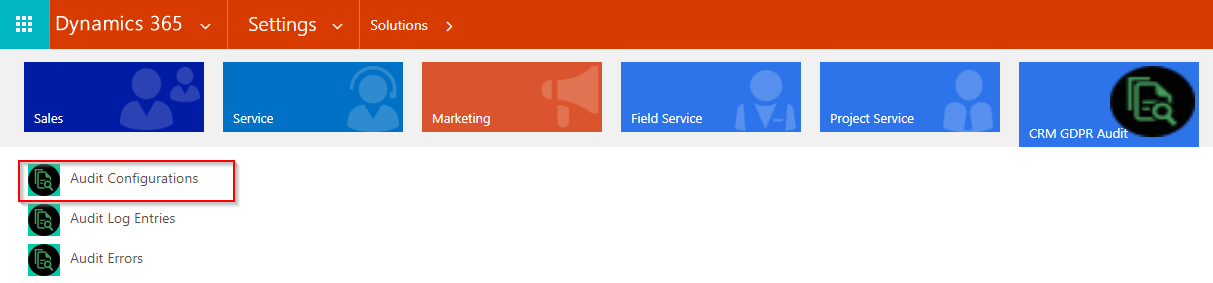
## Installation

Install the solution manually or via the AppSource store. Remember to do a **Publish All Customizations** after you have installed.

If you’re running a Sandbox instance, make sure that background operations (workflows) are enabled. The CRM GDPR Audit solution operates mostly through asynchronous operations.

## Create a basic configuration

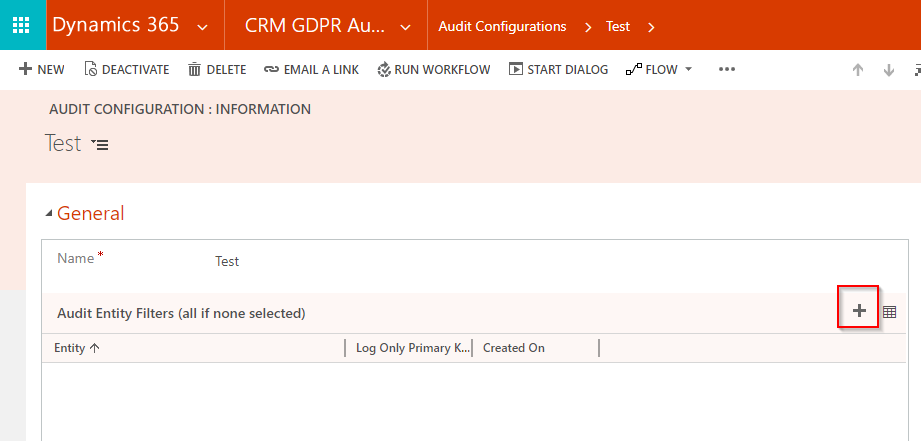
1. In the main navigation, navigate to CRM Gdpr Audit\Audit Configuration and create a new configuration record. If you don’t see the mentioned elements in the sitemap, make sure you have published all customizations and refreshed the page.

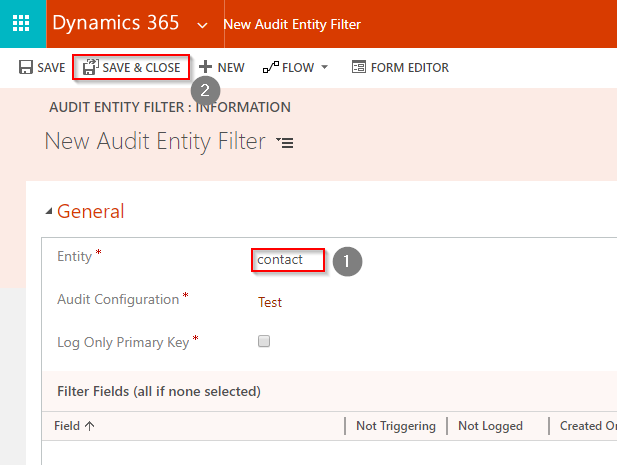




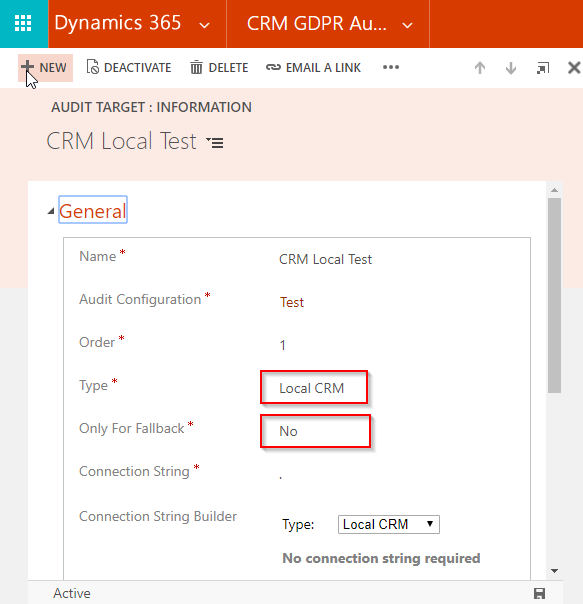
You can put in any value in the *Name* field. It’s only for reference. Save the record.

1. Create a new Audit Entity Filter for the contact entity.





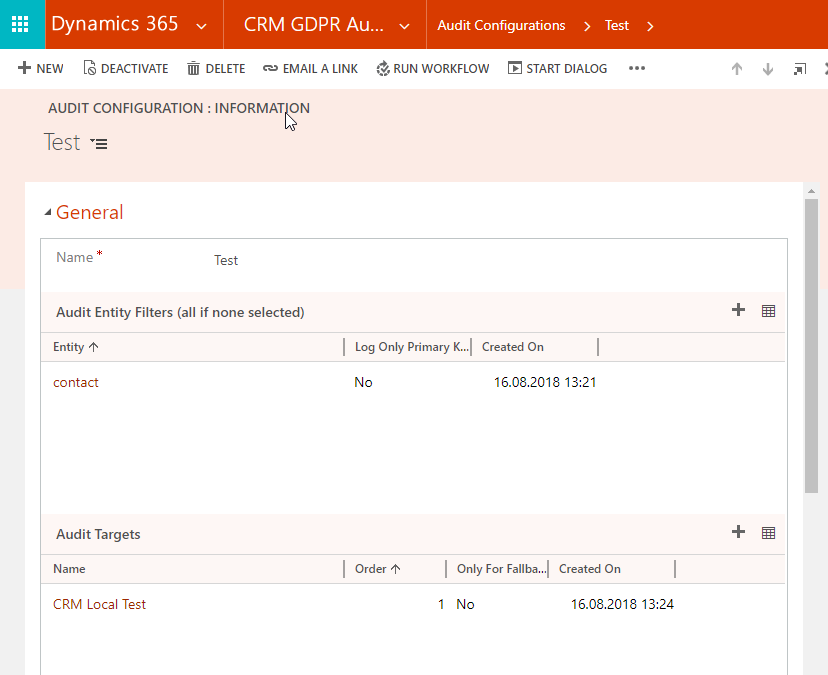
1. The same way create a new Audit Target. Set the *Type* to *Local CRM* and *Only For Fallback* to *No*.



The connection string is required, but it’s value doesn’t matter for the Local CRM Target. You can put any value there.

**At this point the solution should be configured** and audit reads of the contact entity.

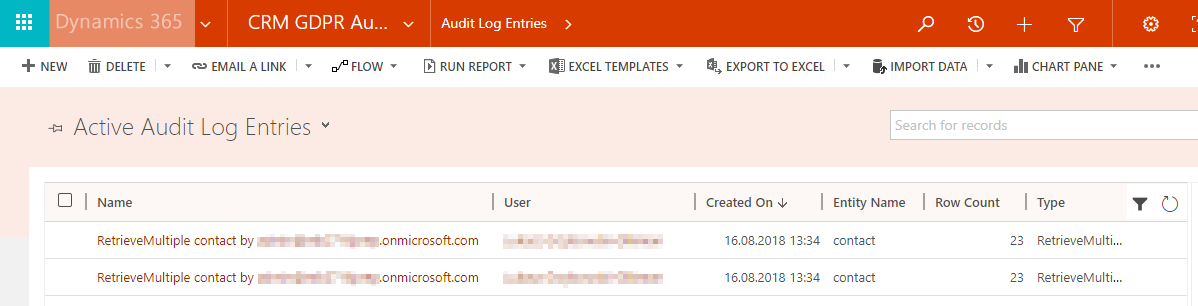
The final configuration should look something like this:



## Testing

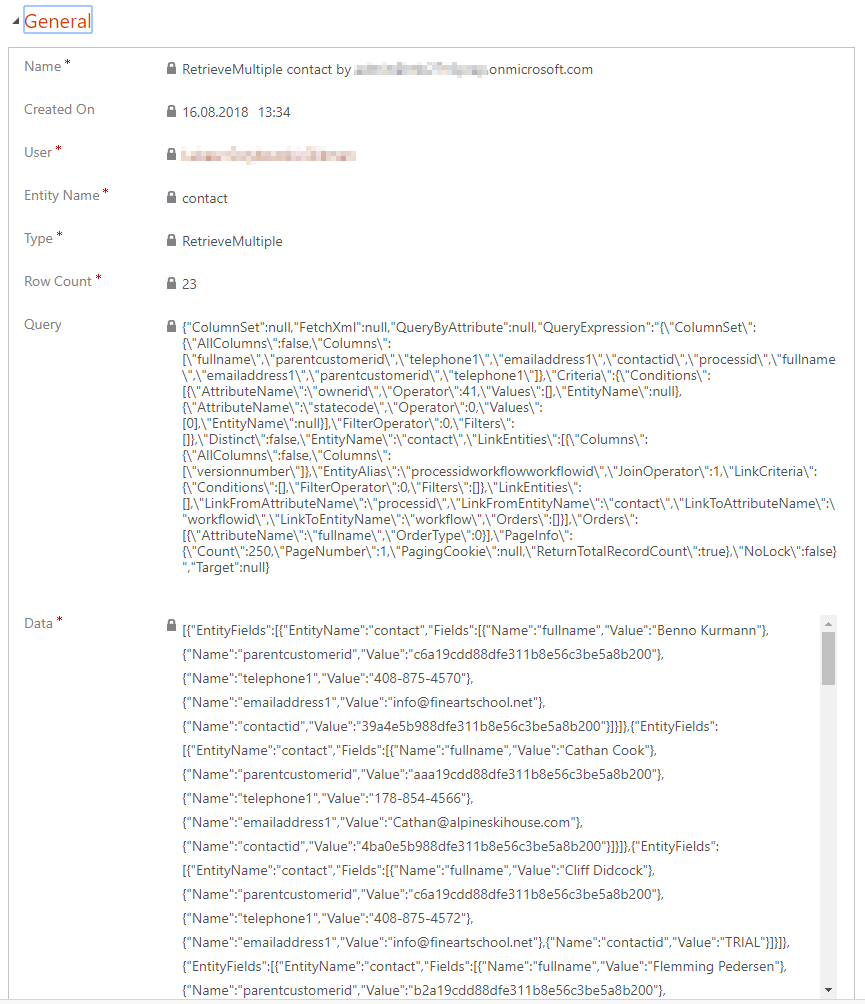
In order to test that the solution works open two tabs (or windows) with Dynamics CRM. In one window open the contact list, in the other navigate to Dynamics 365\CRM GDPR Audit\Audit Log Entries. Try to refresh the contact list a couple of times.

After a moment you should begin seeing audit log entries appear on the list.



If they don’t refer to the Error Handling section of this document.

By opening a log record, you can see the details of the information being logged. The image below is based on standard CRM sample data.



# Error handling

The solution can encounter two kinds of errors:

1. Critical errors
2. Target errors

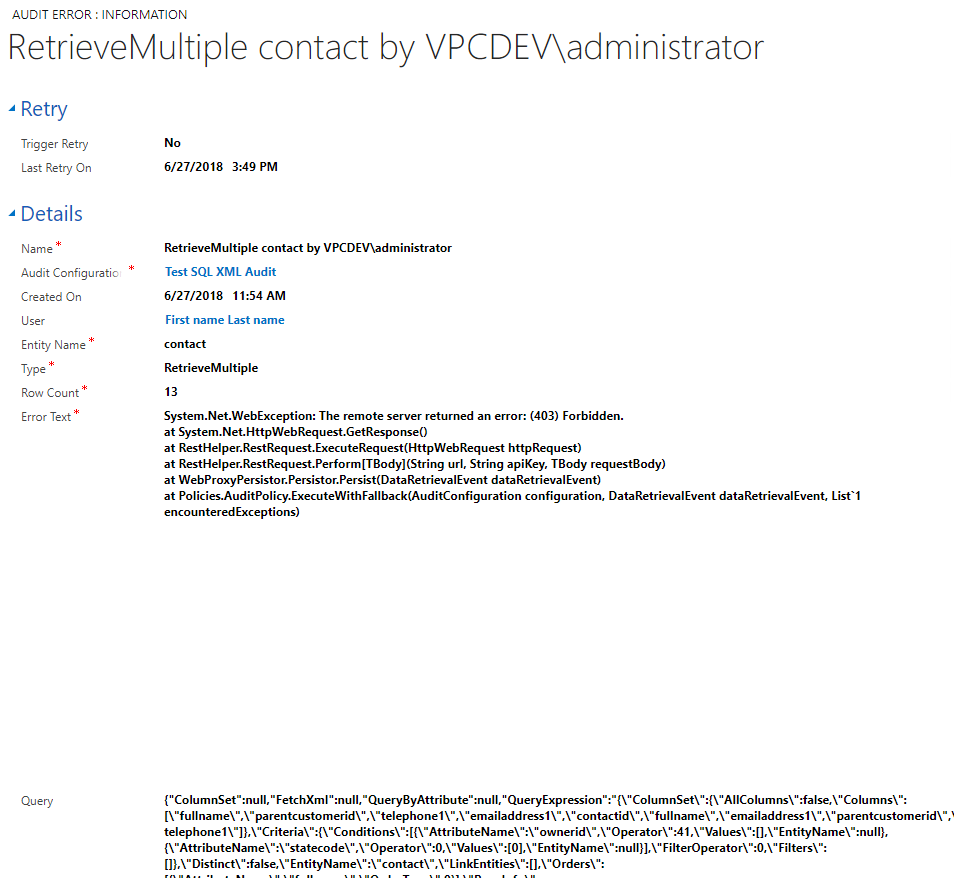
**Critical errors** originate form uncaught exceptions, CRM database errors, Asynchronous Processing Service errors etc. They can be monitored by looking at Setting\System Jobs inside CRM.

In general their names will start with CrmGdprAudit.

**Target errors** will occur if for example the specified target is unavailable. In such case the solution will try logging the data in the fallback targets.

The main difference here is that all normal targets are processed. Even if one of them encounters an error it the process will continue to the next one. For fallback targets processing will stop at the first one that succeeds.

Additionally, whenever an error occurs a new record in the entity Audit Errors will be created. It contains all information about the logged event plus additionally a description of the error.



After the errors with the target are fixed such an error can be reprocessed. This can be done by setting the Trigger Retry value to “true” and saving the record. An asynchronous plugin will run in the background and try to again save the record. The current error will be deactivated. If new error occur, than new Audit Error records will be created.

