Ramco API v 2.0

Updated May 13th, 2021

The most up-to-date version of this document can always be found at
https://api.ramcoams.com/api/v2/ramco_api_v2_doc.pdf

General Overview

Intent is to provide an API that allows for discovery of what can be accessed (entity types, their properties and metadata), formulation of queries that can support custom entities and properties, and record management functions using an easy to understand syntax with a minimum of custom api calls.

User Authentication

Users are validated and granted access by passing in an assigned key value as a post parameter with every request. A separate authentication process is not required.

API Call Process

All requests are posted to https://api.ramcoams.com/api/v2/ (note https, ssl is required).

Post parameters control which operation is performed and return values.

The user's key must be provided with every request and will tie to a specific user profile which specifies the appropriate Ramco backend and permissions level.

Parameter names are case-insensitive. Parameter values, however, may be case-sensitive.

Operation Overview

- **Metadata Operations**
  - **GetEntityTypes:** Returns a complete list of all entities in the system.
  - **GetEntityMetadata:** Returns complete field list for an entity, along with important information about the field (type, length, etc.).
  - **GetOptionSet:** Returns the key/value sets for the specified OptionSet.
  - **ClearCache:** Flushes the server-side cache of metadata.
- **Data Querying Operations:**
  - **GetEntity:** Returns attribute and/or relationship data for a single entity.
  - **GetEntities:** Returns attribute and/or relationship data for 0-N entities.
  - **ValidateUser:** Validate a user based on provided username and password.
- **Data Transformation Operations:**
  - **UpdateEntity:** Allows for update of field-level entity data.
  - **CreateEntity:** Allows for the creation of new entity records.
  - **DeleteEntity:** Allows for the deletion of existing entity records.

Both GetEntity and GetEntities support querying the attributes of the requested entity and related entities (ex: Can return cobalt_membership data with the Contact).

GetEntity and GetEntities differ in how data is requested. GetEntity uses the record guid and returns 0-1 records. GetEntities supports rich query filters and returns 0-N records.

Response Overview

Response format will be a standardized json package across all responses/operations and will contain:

- **ResponseCode:** Numeric code that generally corresponds to http response codes (200 = OK, 204 = Processed, no content, 400 = Bad Request, etc.). This element is always present in the response.
- **ResponseText:** Text that corresponds to responseCode, providing more detail where required. (ex: "UserPhone is invalid field for Contact entity."). This element is always present in the response.
- **Data:** Contains the response data set, if any.
- **StreamToken:** To prevent very large GetEntities requests from consuming huge amounts of memory, the resultset will be returned once a memory threshold has been reached even if all results haven't been returned. If this occurs, the StreamToken will be populated with a token that can be used to fetch the next batch of records.

Ramco Data Types

Dates

All dates – those returned by Ramco and those used in GetEntities queries – are RFC 3339 format and indicate Greenwich Mean Time. Trailing timezone indicator is not supported.
Examples:

- 2013-12-31T18:30:00
- 2013-12-31

Attachments

Files (images, word documents, pdfs, etc.) can be stored as long strings in Ramco. They can be fetched, updated, created like any other data point but they cannot be used in filter criteria. Files are stored Base64Encoded, so they will need to be decoded client-side. When uploading a document to Ramco, it will need to be Base64Encoded prior to upload.

Call Overview

GetEntityTypes

Fetch all entities in the system.

Post params:

- Key = your provided authentication key
- Operation = GetEntityTypes

GetEntityMetadata

Fetch metadata on entity type Contact (includes entity description, attributes and relationships).

Post params:

- Key = your provided authentication key
- Operation = GetEntityMetadata
- Entity = Name of entity being queried (ex: Contact)

GetOptionSet

Fetch the valid value/label pairs for the specified OptionSet (what Ramco calls a picklist)

Post params:

- Key = your provided authentication key
- Operation = GetOptionSet
- Entity = Name of entity being queried (ex: Contact)
- Attribute= Name of the attribute being queried (ex: PreferredPhone)

ClearCache

Clears the server-side metadata cache. If an entity or attribute has been added (or removed), then clearing the cache will permit the changes to be reflected immediately. The cache will normally expire every 24 hours.

Post params:

- Key = your provided authentication key
- Operation = ClearCache

GetEntity

Fetch the attributes and/or relationships of one specific entity using guid.

Post params:

- Key = your provided authentication key
- Operation = GetEntity
- Entity = Type of entity being queried (ex: Contact)
- Guid = guid of entity to return
- Attributes = comma separated list of attributes to return

Attributes for related entities can be also be specified by including 'relationship/attribute' in the attributes parameter. As an example, the metadata for the Contact entity shows a relationship to cobalt_memberships entity named cobalt_contact_cobalt_membership. To have the query return the status of those memberships, include 'cobalt_contact_cobalt_membership/statuscode' in the attribute parameter.

GetEntities

Fetch the attributes and/or relationships of multiple entities of one type using a user-defined filter.

Post params:

- Key = your provided authentication key
- Operation = GetEntities
- Entity = Name of entity being queried (ex: Contact)
- Filter (optional) = User-specified filter string
- Attributes = comma separated list of attributes to return
- StringDelimiter (optional) = User-specified delimiter used to wrap string values (default is #)
- MaxResults (optional) = Maximum number of entities to return

GetEntities Filter syntax

The GetEntities operation supports filtering which allows multiple criteria to be specified using an attribute<operator>value syntax. Multiple filters can be strung together using AND/OR. Nested filters are also supported via parentheses.

Supported operators:

- <eq> = equal to
- <ne> = not equal to
- <gt> = greater than
- <ge> greater than or equal to
- <lt> less than
- <le> less than or equal to
- <sb> string begins with (valid for strings only)
- <sc> string contains (valid for strings only)
- <se> string ends with (valid for strings only)

Strings must be wrapped with a delimiter. The default delimiter is '#' but a user-specified delimiter can be utilized by including the optional StringDelimiter param.

Example filter for records modified on or after Jan 1, 2014, where the LastName field starts with 'S' and the cobalt_EmailVerified is false or null:
'ModifiedOn<ge>2014-01-01 AND LastName<sb>#S# AND (cobalt_EmailVerified<eq>null OR cobalt_EmailVerified<eq>0)'

GetEntities using StreamToken

If the response to a previous GetEntities request includes a StreamToken (see "Response Overview" above), it can be used to continue fetching results. The post params are different in this case.

Post params:

- Key = your provided authentication key
- Operation = GetEntities
- StreamToken = The alphanumeric streamtoken returned from the previous request

ValidateUser

Returns the globally unique id of a contact that matches the provided cobalt_username and cobalt_password parameters. Returns a 422 error when there is no user with provided username/password combination.

Post params:

- Key = your provided authentication key
- Operation = ValidateUser
- cobalt_username = User's username
- cobalt_password = User's password

User's log in to the Ramco CRM with their cobalt_username and cobalt_password credentials. This function allows for the validation of a user based on these values even when the cobalt_password field is salted and hashed. The return value will be the guid of the identified contact record. When no match is found a 422 error will be returned in the response with an error message of "No user with provided username/password combination."

UpdateEntity

Allows for modification of attributes of existing Entities.

Post params:

- Key = your provided authentication key
- Operation = UpdateEntity
- Entity = Type of entity being modified (ex: Contact)
- Guid = guid of entity being modified
- AttributeValues = Comma separated attribute=value pairs.
- StringDelimiter (optional) = User-specified delimiter used to wrap string values (default is #)

The AttributeValue parameter syntax

A comma delimited list of attribute value pairs like:
FirstName=#Joe#,Birthday=1980-12-31,EmailVerified=true,NumChildren=3

Strings must be wrapped with a delimiter. The default delimiter is '#' but a user-specified delimiter can be utilized by including the optional StringDelimiter param.

OptionSets must be set to a numeric value valid for that optionset, or zero to clear current existing value. To determine values that are valid for an optionset, use the GetOptionSet api call.

Note that when updating an attribute of type EntityReferene, usually only the guid needs to be specified. In cases where the EntityReference can be of more than one type (ex: OwnerId can be SystemUser or Team) the value must be type:guid (SystemUser:fb50333e-6b9f-e111-8d5d-00155d000140).

In instances where the string delimiter occurs in the string itself (ex: 'Some#Text'), the delimiter must be Base64Encoded (ex: 'SomeIw==Text'). Of course the option to change the delimiter using the StringDelimiter parameter exists as well.

CreateEntity

Allows for creation of a new entity record.

Post params:

- Key = your provided authentication key
- Operation = CreateEntity
- Entity = Type of entity being modified (ex: Contact)
- AttributeValues = Comma separated attribute=value pairs.
- StringDelimiter (optional) = User-specified delimiter used to wrap string values (default is #)

The AttributeValue parameter syntax

The AttributeValue parameter for CreateEntity follows the same format as that for UpdateEntity.

PrimaryIdAttribute

Every entity type has a PrimaryIdAttribute which can be determined with a GetEntityMetadata call. If a specific guid is desired for the entity being created, it can be added to the AttributeValues parameter along with the other specified values. If the PrimaryIdAttribute is not specified, one will be automatically generated.

Example creating a Contact record (ContactId is PrimaryIdAttribute):
FirstName=#John#,LastName=#Doe#,ContactId= 84a4d17f-2e48-4a4b-bb28-26f7c14fc926, Birthday=1980-12-31

DeleteEntity

Allows for the deletion of existing entity records.

Post params:

- Key = your provided authentication key
- Operation = DeleteEntity
- Entity = Type of entity being queried (ex: Contact)
- Guid = guid of entity to delete

Valid Reponse Codes

Code	Description:Short	Desciption:Verbose
200	OK	The request was successfully processed and data is included in the response.
204	OK: No Data	The request was successfully processed but no data is included in the response. This is typical of UpdateEntity requests.
206	OK: Partial Data	The request was successfully processed and partial data is included in the response. This is the expected response when the dataset that Ramco needs to return to the user is too large. A StreamToken will be returned to allow the user to fetch the remaining data.
400	Bad Request	The request was not understood. See the response text for more information.
401	Unauthorized	The request was understood but it will not be fulfilled due to a lack of user permissions. See the response text for more information.
404	Not Found	The request is understood but no matching data is found to return.
422	Invalid User	No user with provided username/password combination. This error is specific to the AuthenticateUser request.
500	Server Error	Something isn't working correctly server-side. This is not an issue that can be resolved by modifying query syntax.

High-Value Data

The primary use of the Ramco API has been to query for new and updated records and then to pull that data down for other systems. To that end, it's useful to know where the data for people and offices resides and to know how to identify new and updated records.

People exist in Ramco as contact records. The contact record will store the vast majority data relevant to people, including address, phone numbers, email address and website data.

A contact record will not be sufficient to determine if someone is a member, staff or vendor with any association, though; that data is stored in cobalt_membership records. These cobalt_membership records will store the membership type (ex: REALTOR®, Staff, Vendor, Affilaite), the status (ex: Active, Inactive), start and end dates and other information that would be specific to a persons relationship with an association.

Office data is stored in Account records. Similar to Contact records for people, the Account record will store data that's specific to an office like their Name, physical and mailing addresses and NRDS ID.

Every record in Ramco will have a CreatedOn and a ModifiedOn datetime field. The ModifiedOn is most useful because it updates both upon record creation and upon modification. It's common to query for modified records on a regular basis to identify those records that have changed since last processing. Do note that if you need to identify changes to people and their statuses (active REALTOR®, etc.) it's important that you check both the ModifiedOn timestamps from the Contact record AND the cobalt_membership records. This is required because some updates (lastname, phone numbers, etc.) are stored in the Contact record and some updates (membership status, type, etc.) are stored in cobalt_membership records. Failure to check all appropriate record types for changes may result in an incomplete or stale picture of a person's status in Ramco.