# Eloqua Bulk API Documentation

Oracle Corporation

May 23, 2014

CONTENTS

CHAPTER

ONE

# INTRODUCTION TO ELOQUA'S BULK API

Eloqua's Bulk API enables you to import and export contact, activity, account, and custom object data into and out of Eloqua at scale. The Bulk API uses the same general pattern for all calls: once you are comfortable with the workflow, you're good to go.

The Bulk API is *scalable*, working efficiently on large data sets and in multi-tenant systems. Its design provides straightforward setup, a simple and consistent interface using REST principles, and it provides granular feedback for troubleshooting.

The following sections provide the knowledge you need to start working with the Bulk API.

## 1.1 Getting Started with the Bulk API

While the Bulk API is designed for ease of use, there are things to bear in mind when determining how you will interact with the Bulk API.

### 1.1.1 Requirements

The Bulk API is designed to let developers start developing with minimal setup or configuration effort. At a minimum, you need an Eloqua instance, and an account on that instance. In order to be able to interact with your Eloqua data, your account needs adequate permissions to access contact fields, secondary assets, etc.

Some Eloqua instances enable contact-level security, which restricts access to data based on different user roles. Users might only have access to contacts located in their geographical region, for instance. Because these permissions affect what data the user can access, it is important that the Eloqua user accessing the API have the appropriate permissions.

### 1.1.2 Considerations

#### Use Case

The Bulk API does not constrain how you interact with it, but your use case should determine how you develop your integration.

For example, if you are importing data into Eloqua, you should consider *resiliency*: if an item fails to import, can you send it again in 15 minutes, or do you need to deal with the failure immediately? Your use case will determine what measures you need to put into place to deal with this.

Similarly, if you are exporting data from Eloqua, how large a data set are you working with? Larger exports will be time consuming, so you can break the export up into smaller chunks using a `filter`. Be aware of the data volumes you expect to deal with and create your export definitions accordingly.

The *Troubleshooting* (page 6) section discusses ways you can monitor your imports and exports and leverage this data to improve your integration.

#### Authentication and Security

Eloqua uses SSL with 128-bit encryption to securely transmit traffic in all API calls.

For authentication, Eloqua's Bulk API supports HTTP Basic Authentication as well as OAuth 2.0.

OAuth is the preferred method of authenticating. OAuth allows Bulk API-powered applications to access resources on behalf of a resource owner without needing the resource owner's credentials.

The Bulk API supports 2-legged OAuth for second party apps, and 3-legged OAuth for apps in the AppCloud™.

In cases where implementing OAuth is not feasible, you can use HTTP Basic Authentication. This approach requires that you obtain the user's credentials, which is not ideal. For example, if a user changes their password, you will need to obtain the new password. If feasible, use OAuth over basic authentication.

The *Authenticate Using OAuth* (page 22) and *Authenticate using HTTP Basic Authentication* (page 26) tutorials provide step-by-step instructions for authenticating to Eloqua using each method.

### 1.1.3 Accessible Elements

The Bulk API is gives you access to contacts, accounts, custom data objects, email groups, and external activities. See: *Introduction to Eloqua Elements* (page 5) for a full description of each element.

### 1.1.4 API Call Format

Interactions with the Bulk API follow a consistent pattern: you create a record definition that tells Eloqua what data you are importing or exporting, move that data into a staging area, and either retrieve the data, in the case of an export, or push that data into Eloqua, in the case of an import.

The *API Call Format* (page 2) document explores this pattern in detail, touching on the rationale behind the three-step workflow. It also discusses data formatting requirements, header requirements, and so on.

You should familiarize yourself with the API call format before starting to work with the Bulk API.

## 1.2 API Call Format

All Bulk API calls use the same general call pattern: whether you are importing or exporting data, the structure of your call will be consistent.

### 1.2.1 Pattern

Importing and Exporting data with the Bulk API follows a three-step pattern:

1. Define the import or export. The definition describes how Eloqua's fields map to your own. For example, your 'email' field might map to Eloqua's `Contact.Field(C_EmailAddress)`. You can also give the definition a name, so that you can reuse it later, specify how to filter the data, and define any additional actions you want to perform. The import or export definition tells Eloqua what kind of operation to prepare to perform.
2. Move data into the staging area. For Imports, this means `POST`-ing your data to the staging area; for Exports, this means telling Eloqua to move its data into the staging area. The staging area is a middle ground that allows read and write operations to occur asynchronously from the HTTP requests you send.

3. Move the data to its final destination. For Imports, this means telling Eloqua to sync the data from the staging area into its database; for Exports, this involves retrieving the data from the staging area.

This design allows for fast client requests, while longer actions are performed asynchronously. By not interacting with the Eloqua database during the HTTP request, you gain a consistent expectation of how long I/O operations will take. The asynchronous pattern also enables scalability: were I/O operations and merges performed inline, database locks would impact the performance of your Eloqua instance.

The staging area system allows data to be written to disk, and then processed in the background without affecting performance.

### 1.2.2 Basic Structure

Familiarizing yourself with the common URI parameters, required HTML headers, and JSON patterns will give you a strong foundation from which to start using the Bulk API.

#### Call URL

The URL you need to call to access Eloqua's Bulk API depends on which "pod" your Eloqua instance is hosted on. The base url is: `https://<host>/api/bulk/2.0`, where `<host>` can be `secure.p01.eloqua.com`, `secure.p02.eloqua.com`, or `secure.p03.eloqua.com`. To find your URL, see: Determining Endpoint URLs on Topliners.

#### URL Parameters

The Bulk API's HTTP `GET` endpoints support a variety of URL parameters that you can use to control what data you retrieve.

- `limit` : specifies the maximum number of records to return
- `offset`: specifies an offset that allows you to retrieve the next batch of records. For example, if your `limit` is 1000, specifying an `offset` of 1000 will return records 1000 through 2000.
- `q`: specifies query criteria used to filter results.

  The `q` format is `<term><operator><value>`. For example: `name='Email*''`.
- `orderBy`: specifies the name of the property to order the results by.

  The `orderBy` format is `<term> ASC | DESC`. For example, `orderBy=name ASC`.

#### HTML Headers

Eloqua's Bulk API supports most common HTML Headers: in addition to the authentication headers for users of *Basic Authentication* (page 26), the `Content-Type` and `Accept` headers specify data formats for the data you import into Eloqua and the Bulk API's response data.

##### Content-Type

The Bulk API supports both JSON and CSV file formats as data sources for `PUT` and `POST` requests. The Bulk API **does not** support XML.

For `PUT` and `POST` requests, you **must** specify either `application/json` or `text/csv` in the `Content-Type` header: if you do not include the Content-Type header an error will occur. The Content-Type header is not required for `GET` and `DELETE` requests, since these do not accept data inputs.

### Accept

The `Accept` parameter specifies what file formats you, the client, are willing to accept from the server: either JSON or CSV. Use of the Accept header is optional: if you do not specify a parameter for `Accept`, the response will default to JSON.

The following call requests the list of Contact fields in CSV format:

```
GET https://<host>.eloqua.com/api/bulk/2.0/contacts/fields
Accept: text/csv
```

The response would resemble:

```
name,internalName,dataType,defaultValue,hasReadOnlyConstraint,hasNotNullConstraint
Email Address,C_EmailAddress,emailAddress,,False,False
First Name,C_FirstName,string,,False,False
```

The following requests the list of Contact fields in JSON format:

```
GET https://<host>.eloqua.com/api/bulk/2.0/contacts/fields
Accept: application/json
```

The response resembles:


```
{
    "count": 82,
    "hasMore": false,
    "items": [
        {
            "createdAt": "1900-01-01T05:00:00.0000000Z",
            "dataType": "emailAddress",
            "hasNotNullConstraint": false,
            "hasReadOnlyConstraint": false,
            "hasUniquenessConstraint": true,
            "internalName": "C_EmailAddress",
            "name": "Email Address",
            "statement": "{{Contact.Field(C_EmailAddress)}}",
            "updatedAt": "1900-01-01T05:00:00.0000000Z",
            "uri": "/contacts/fields/100001"
        },
        {
            "createdAt": "1900-01-01T05:00:00.0000000Z",
            "dataType": "string",
            "hasNotNullConstraint": false,
            "hasReadOnlyConstraint": false,
            "hasUniquenessConstraint": false,
            "internalName": "C_FirstName",
            "name": "First Name",
            "statement": "{{Contact.Field(C_FirstName)}}",
            "updatedAt": "2013-03-26T21:32:13.2530000Z",
            "updatedBy": "Docs.Example",
            "uri": "/contacts/fields/100002"
        }
    ]
}
```

### 1.2.3 Example Call & Response

You can create Bulk API requests using whatever language you wish. The examples and tutorials in this guide show the basic request that you need to send to perform your request, ignoring language-specific code examples.

Request:

```
https://<host>.eloqua.com/API/Bulk/2.0/contact/import/123
```

Response:

```
{
    "createdAt": "2014-05-13T18:25:32.0270000Z",
    "createdBy": "Allison.Moore",
    "fields": {
        "C_EmailAddress": "{{Contact.Field(C_EmailAddress)}}",
        "C_FirstName": "{{Contact.Field(C_FirstName)}}",
        "C_LastName": "{{Contact.Field(C_LastName)}}"
    },
    "identifierFieldName": "C_EmailAddress",
    "isSyncTriggeredOnImport": false,
    "isUpdatingMultipleMatchedRecords": false,
    "name": "Docs Import Example",
    "syncActions": [],
    "updatedAt": "2014-05-13T18:25:32.0270000Z",
    "updatedBy": "Allison.Moore",
    "uri": "/contacts/imports/1183"
}
```

**See also:**

The *Import Data Into Eloqua* (page 9) and *Export Data from Eloqua* (page 12) tutorials provide step-by-step instructions for importing and exporting data with the Bulk API.

## 1.3 Introduction to Eloqua Elements

The Bulk API enables you to import and export data for numerous elements within Eloqua. The Bulk API also has its own elements that it uses to interact with Eloqua.

The following sections describe all of the elements encountered when using the Bulk API, and highlights the different interaction patterns associated with each type.

### 1.3.1 Eloqua Elements

The "Eloqua Elements" are elements stored in Eloqua. Each element type has its own set of associated fields.

The Bulk API exists to allow you to import and export data related to these elements. However, due to the *asynchronous design* (page 2) of the Bulk API, you never interact directly with the Eloqua Elements. Rather, you move data into a *staging area* and then synchronize the data into Eloqua or out for your use.

The Eloqua elements are:

- Contacts
- Accounts
- Custom Data Objects

- Email Groups
- Activities: user-uploaded data, often from events, hospitality, webinars, or email campaigns

With the exception of Activities, you can use a `/fields` endpoint to retrieve the list of fields associated with each element in your Eloqua instance. See: *Search for Field Names* (page 15) for more information.

Activities behave different from the other Eloqua elements. Each **Activity Type** has its own set of fields. Refer to *Activity Fields* (page 38) for a full description of the Activity types that the Bulk API supports and their specific fields.

### 1.3.2 Bulk API Elements

The "Bulk API Elements" are objects that you create in order to interact with Eloqua through the Bulk API. The are, essentially, metadata elements: elements that provide context to Eloqua and to the Bulk API to allow them to perform the appropriate actions.

These bulk elements are:

- Imports
- Exports
- Syncs

Unlike the "Eloqua Elements", which you interact with indirectly, the Bulk API enables you to perform direct CRUD operations on the metadata. If you `GET` an import or an export, Eloqua returns its data. In contrast, you cannot directly `GET` contact or activity data: you have to go through the export steps to obtain it.

## 1.4 Troubleshooting

Eloqua's Bulk API includes endpoints to help you troubleshoot your imports and exports. Specifically, these endpoints provide insight into what happened during the sync phase of an import or export.

---

**Note:** In order to control for different data access permissions, Eloqua associates imports and exports with the user who creates them. Thus, only the user who synchronizes an export is able to retrieve the exported data: any other user who attempts to retrieve it will receive an error.

If you wish to retrieve data for a given export definition, you will need to re-sync that data yourself, and then retrieve it.

---

### 1.4.1 View Sync Logs

Sync Logs provide aggregate data detailing what happened during a specific sync. Unlike the sync's `status` field which simply indicates success, failure, and errors, the `/syncs/{id}/logs` endpoint response includes the *HTTP Status Codes* (page 43) and a human-readable message.

For example, calling `https://<host>.eloqua.com/API/Bulk/2.0/syncs/1196/logs` yields:

```
{
    "count": 3,
    "hasMore": false,
    "items": [
        {
            "count": 4149,
            "createdAt": "2014-05-12T14:23:05.9100000Z",
            "message": "Successfully exported members to csv file.",
```

```
            "severity": "information",
            "statusCode": "ELQ-00102",
            "syncUri": "/syncs/1196"
        },
        {
            "count": 0,
            "createdAt": "2014-05-12T14:23:07.2670000Z",
            "message": "Deleted internal staging for data transfer.",
            "severity": "information",
            "statusCode": "ELQ-00131",
            "syncUri": "/syncs/1196"
        },
        {
            "count": 0,
            "createdAt": "2014-05-12T14:23:07.7630000Z",
            "message": "Successfully converted csv file to sqlite, taking 264 kb.",
            "severity": "information",
            "statusCode": "ELQ-00106",
            "syncUri": "/syncs/1196"
        }
    ],
    "limit": 1000,
    "offset": 0,
    "totalResults": 3
}
```


This provides more granular information and can help you debug where an unsuccessful sync went wrong.

### 1.4.2 Check for Import Errors

The `/syncs/{id}/rejects` endpoint gives raw data about validation failures during **imports**.

Eloqua performs validation on some data. If you attempt to import a non-valid email address using the Bulk API, that record will be rejected. The `/syncs/{id}/rejects` endpoint allows you to see what records were not imported, so that you can correct any issues and try again.

### 1.4.3 Retrieve Past Data Files

In some cases, you may choose to reuse an export definition: if you are performing routine exports, for example, reusing an existing definition saves time and effort. However, when you re-sync the export, the data exported necessarily changes. Last week's output from `/contacts/exports/123/data` may be different from today's.

The `/syncs/{id}/data` endpoint enables you to retrieve the data associated with a *specific sync*. This can be useful to retrieve past data or debug issues retroactively.

CHAPTER

TWO

# TUTORIALS

## 2.1 Import Data Into Eloqua

**Important:** If you are importing a large volume of data, break your export up into smaller chunks. This will help mitigate performance problems on your Eloqua instance and help you to avoid reaching the API limits.

Import operations with Eloqua's Bulk API follow a general pattern:

1. Create the import definition. The import definition describes how Eloqua's fields map to the data you are importing. This is also where you can specify any *Sync Actions* (page 40) that Eloqua should perform when importing the data.
2. Push your data into the staging area. The Bulk API supports JSON and CSV file formats.
3. Synchronize the data from the staging area into Eloqua.

When you import data, Eloqua performs an `UPSERT` operation. An `UPSERT` checks if a record already exists in Eloqua and updates it with the new data if it does exist, or creates a brand new record containing the inserted data if there is not already a record in the database.

Eloqua's Bulk API supports importing *contacts*, *accounts*, *custom objects*, and *external activities*. While the types of data you are importing may vary, the import workflow remains the same.

### 2.1.1 Create the Import Definition

An import definition includes what type of data you are importing, defines how the external data maps to Eloqua's entity fields, and chooses the *identifier field* that you will use to check the imported data against existing records within the system. The import definition can also include information about *sync actions* (page 40) and configuration options for the import. The import reference describes the valid parameters.

1. Choose what type of data you will import: contact data, account data, custom object data, or external activity data. Each element type has different associated fields, so you need to know what kind of data you are importing before you can determine what fields you will map your data to.

   The sample data is Contact data, and resembles the following:

```
{
   {
      "firstName" : "Juan",
      "lastName" : "Garcia",
      "emailAddress" : "juan@example.com"
   },
   {
      "firstName" : "Tatiana",
```

```
        "lastName" : "Smirnov",
        "emailAddress" : "tatiana@example.com"
    }
}
```

2. Determine the field mapping. Field mappings describe how external data maps to Eloqua's entity fields. The Bulk API uses *Eloqua Markup Language (EML)* (page 29) statements to define mappings between external entities and Eloqua entities.

   To determine the entity fields within Eloqua that you will need, as well as the appropriate EML statement, the Bulk API provides `/fields` endpoints: for detailed instructions on searching for field names, see the *Search for Field Names* (page 15) tutorial.

3. Optionally, specify *sync actions* (page 40). Sync actions are actions performed when the Bulk API syncs your data into Eloqua. For example, adding or removing a contact from a campaign, or changing a contact's status in an email group from "unsubscribed" to "subscribed".

4. Send the request:

   ---

   **Note:** The URL you need to call to access Eloqua's Bulk API depends on which "pod" your Eloqua instance is hosted on. The base url is: `https://<host>.eloqua.com/api/bulk/2.0`, where `<host>` can be `secure.p01`, `secure.p02`, or `secure.p03`. To find your URL, see: Determining Endpoint URLs on Topliners.

   ---

```
POST https://<host>.eloqua.com/api/bulk/2.0/contacts/imports/

{
  "name": "Docs Import Example",
  "fields": {
      "firstName": "{{Contact.Field(C_FirstName)}}",
      "lastName": "{{Contact.Field(C_LastName)}}",
      "emailAddress": "{{Contact.Field(C_EmailAddress)}}"
   },
   "identifierFieldName": "emailAddress",
   "isSyncTriggeredOnImport" : "false"
}
```

   The import definition **must** include the `fields` and `identifierFieldName` parameters.

   The import definition maps the source data's `firstName`, `lastName`, and `emailAddress` fields to Eloqua's `{{Contact.Field(C_FirstName)}}`, `{{Contacts.Field(C_LastName))}}`, and `{{Contacts.Field(C_EmailAddress)}}` fields.

   The `identifierFieldName` specifies what field Eloqua should use to match your data to the data in Eloqua. Choose a field that is likely to be **unique** to avoid updating the wrong record.

   Any other field is an "import characteristic". For the full list of import characteristics, see: *Import Characteristics* (page 39)

   The `name` parameter describes the import definition, to facilitate reuse and locating the import in the future.

   `isSyncTriggeredOnImport` is an important parameter: if set to `true`, the Bulk API automatically syncs your data into Eloqua when you upload it to the staging area. If set to `false`, you must manually create the sync operation that merges your data into Eloqua. Manually syncing the data provides more control over the timing of the syncs, and allows you to break large sync operations into smaller batches. This can mitigate performance issues in your Eloqua instance. By default, `isSyncTriggeredOnImport` is set to `true`.

5. Receive the response. If successful, the response should resemble:

```
{
    "name": "Docs Import Example",
    "fields": {
        "firstName": "{{Contact.Field(C_FirstName)}}",
        "lastName": "{{Contact.Field(C_LastName)}}",
        "emailAddress": "{{Contact.Field(C_EmailAddress)}}"
    },
    "identifierFieldName": "emailAddress",
    "isSyncTriggeredOnImport": false,
    "isUpdatingMultipleMatchedRecords": false,
    "uri": "/contacts/imports/1182",
    "createdBy": "Docs.Example",
    "createdAt": "2014-05-13T14:13:30.0402961Z",
    "updatedBy": "Docs.Example",
    "updatedAt": "2014-05-13T14:13:30.0402961Z"
}
```

You will use the returned `uri` parameter to send your data into Eloqua.

### 2.1.2 Push Data to Staging Area

The `/imports/{id}/data` endpoints exist to push data into Eloqua. The Bulk API supports JSON and CSV-formatted data sources. Specify your data format in the `Content-Type` HTML header.

The request should resemble:

```
POST https://<host>.eloqua.com/api/bulk/2.0/contacts/imports/1182/data
Content-Type: application/json

{
    "item": [
        {
            "firstName": "Juan",
            "lastName": "Garcia",
            "emailAddress": "juan@example.com"
        },
        {
            "firstName": "Tatiana",
            "lastName": "Smirnov",
            "emailAddress": "tatiana@example.com"
        }
    ]
}
```

The response, if successful is an `HTTP 204 No Content` message. The data is now in the staging area, ready to be synced into Eloqua.

### 2.1.3 Synchronize the Imported Data

The last step is to synchronize the data from the staging area into the Eloqua database. If the import `isSyncTriggeredOnImport` import characteristic is set to `true`, the sync occurs automatically. If `isSyncTriggeredOnImport` is set to false, you must manually create the sync.

The following is an example of how to sync the data from the import into the Eloqua database. To sync the import with uri `/contacts/imports/1182`:

```
POST https://<host>.eloqua.com/api/bulk/2.0/syncs
```

```
{
   "syncedInstanceUri": "/contacts/imports/1182"
}
```

The response to the above HTTP request will include a uri for the sync, as in the following:

```
{
   "syncedInstanceUri": "/contacts/imports/1182",
   "status": "pending",
   "createdAt": "2014-05-13T17:58:34.0176959Z",
   "createdBy": "Docs.Example",
   "uri": "/syncs/1208"
}
```

Then, to check the status of the sync, perform a `GET` on the sync's URI.

For example:

```
GET https://<host>.eloqua.com/api/bulk/2.0/syncs/1208
```

Yields:

```
{
   "syncedInstanceUri": "/contacts/imports/1182",
   "syncStartedAt": "2014-05-13T17:58:34.2770000Z",
   "status": "success",
   "createdAt": "2014-05-13T17:58:33.8130000Z",
   "createdBy": "Docs.Example",
   "uri": "/syncs/1208"
}
```

**See also:**

If you have problems with your import, refer to the *Troubleshooting* (page 6) document for ways you can debug the issue.

## 2.2 Export Data from Eloqua

---

**Important:** If you are exporting a large volume of data, break your export up into smaller chunks. This will help mitigate performance problems on your Eloqua instance and help you to avoid reaching the API limits.

---

Exporting data from Eloqua using the Bulk API follows the same pattern as data imports:

1. Create the export definition. The export definition describes how Eloqua's fields map to the output fields you require. This is also where you can specify if and how you want to filter the data.
2. Synchronize the data for export using the export URI provided in the export definition response.
3. Retrieve the exported data.

### 2.2.1 Create the Export Definition

---

**Note:** The URL you need to call to access Eloqua's Bulk API depends on which "pod" your Eloqua instance is hosted

on. The base url is: `https://<host>.eloqua.com/api/bulk/2.0`, where `<host>` can be `secure.p01`, `secure.p02`, or `secure.p03`. To find your URL, see: Determining Endpoint URLs on Topliners.

---

An export definition includes what data you are exporting and defines how Eloqua data maps to your fields. The export definition can also include information about *sync actions* (page 40) and configuration options for the export. The export reference describes the full list of valid parameters.

### Request

The following export definition requests activity data and filters it based on the Activity Type, in this case, `FormSubmit`.

```
POST https://<host>.eloqua.com/api/bulk/2.0/activities/exports
```

```
{
   "name":"Example Activity Export",
   "fields":{
      "ActivityId":"{{Activity.Id}}",
      "AssetName":"{{Activity.Asset.Name}}",
      "ActivityType":"{{Activity.Type}}",
      "Id":"{{Activity.Id}}",
      "ActivityDate":"{{Activity.CreatedAt}}",
      "EmailAddress":"{{Activity.Field(EmailAddress)}}",
      "ContactId":"{{Activity.Contact.Id}}",
      "VisitorId":"{{Activity.Visitor.Id}}",
      "AssetType":"{{Activity.Asset.Type}}",
      "AssetId":"{{Activity.Asset.Id}}",
      "RawData":"{{Activity.Field(RawData)}}"
   },
   "filter":"'{{Activity.Type}}'='FormSubmit'"
}
```

The `fields` map Eloqua's terminology to the terminology to use in the exported data: essentially, you are telling Eloqua what to call the exported fields so that the data is usable in your other applications. To see the list of fields for Activities, see: *Activity Fields* (page 38). For other entity types, see: *Search for Field Names* (page 15).

The `filter` parameter enables you to choose which fields to export based on criteria. The `filter` parameter is **optional**. However, you should use filters to ensure that you retrieve the correct data, and to avoid creating excessively large exports that can cause performance problems.

The `name` parameter describes the export definition, to facilitate reuse and locating the export in the future.

### Response

Provided the request was successful, the response will be a `201 Created` that includes the Bulk Export's `name`, `fields` and `filter` fields, along with the new `uri` field, the name of the user who created and updated it, and timestamps for the creation and update:

```
{
   "name":"ATD - Example Activity Export",
   "fields":{
      "ActivityId":"{{Activity.Id}}",
      "AssetName":"{{Activity.Asset.Name}}",
      "ActivityType":"{{Activity.Type}}",
      "ActivityId":"{{Activity.Id}}",
      "ActivityDate":"{{Activity.CreatedAt}}",
```

```
        "EmailAddress":"{{Activity.Field(EmailAddress)}}",
        "ContactId":"{{Activity.Contact.Id}}",
        "VisitorId":"{{Activity.Visitor.Id}}",
        "AssetType":"{{Activity.Asset.Type}}",
        "AssetId":"{{Activity.Asset.Id}}",
    },
    "filter":"'{{Activity.Type}}'='FormSubmit'",
    "uri":"/activities/exports/1",
    "createdBy":"Docs.Example",
    "createdAt":"2014-01-28T21:18:06.5184689Z",
    "updatedBy":"Docs.Example",
    "updatedAt":"2014-01-28T21:18:06.5184689Z"
}
```

The `uri` field is particularly important, as you will use it to synchronize the data for export and to retrieve the data when the export is complete.

### 2.2.2 Synchronize the Data for Export

Once you have set up created the export's record definition, you need to synchronize the data using the `/syncs` endpoint and the export's `uri`.

In this example, the request is:

```
POST https://<host>.eloqua.com/api/bulk/2.0/syncs
```

```
{
    "syncedInstanceUri" : "/activities/exports/1"
}
```

The response will be a `201 Created` and include a `status` field that reflects the status of the export, creation metadata, and the sync `uri`:

```
{
    "syncedInstanceUri":"/activities/exports/1",
    "status":"pending",
    "createdAt":"2014-01-28T21:18:07.5792689Z",
    "createdBy":"Docs.Example",
    "uri":"/syncs/123"
}
```

The sync `status` is `pending`. As the sync progresses, the `status` message will change. To check the status of the sync, you can use the sync `uri` with a GET request:

```
GET htt[s://<host>.eloqua.com/api/bulk/2.0/syncs/123
```

When the sync is complete the response will be a `200 OK` that resembles the following:

```
{
    "syncedInstanceUri":"/activities/exports/1",
    "syncStartedAt":"2014-01-28T21:18:27.3600000Z",
    "status":"success",
    "createdAt":"2014-01-28T21:18:07.5730000Z",
    "createdBy":"Docs.Example",
    "uri":"/syncs/123"
}
```

### 2.2.3 Retrieve the Exported Data

Finally, you can retrieve the data you have exported using the export's `uri`, in this case `/activities/exports/1` and the `/data` endpoint:

```
GET /api/bulk/2.0/activities/exports/1/data
Accept: application/json
```

The response will include the data in the `items` field:


```
{
    "Count": 1,
    "hasMore": false,
    "limit": 1000,
    "offset": 1,
    "totalResults": 1,
    "items": [
        {
            "ActivityId": "999",
            "AssetName": "Forms-BulkActivity",
            "ActivityType": "FormSubmit",
            "ActivityDate": "2014-01-29 13:45:31",
            "EmailAddress": "allison@example.com",
            "ContactId": "123",
            "VisitorId": "1",
            "AssetType": "Form",
            "AssetId": "1",
        }
    ]
}
```


If you do not specify a `limit` when you send your `GET` request, the `limit` defaults to 1000. You can specify any limit up to 50000, though requesting larger volumes may create performance issues.

If there were more than 1000 matching items, `hasMore` would be `true` and you would be able to request the next batch of items using an appropriate `offset`. For example, if there were 1500 items, the following call would request items 1000 through 1500:

```
GET https://<host>.eloqua.com/api/bulk/2.0/contacts/exports/8/data?offset=1000
```

See: *URL Parameters* (page 3) for a full list of parameters you can use to modify what data you request.

## 2.3 Search for Field Names

Every field in Eloqua has two names: the *Display Name* and the *Internal Name*. The Display Name is the name that Eloqua users see when they are editing a contact form; the Internal Name is a unique name for that field. For example, Contacts and accounts both have email address fields: Eloqua may use "Email Address" as the display name for both entity types, but they have distinct Internal Names.

To narrow down the list of parameters to ones that are relevant for you, use query parameters to search for the fields that you need.

For example, the following request searches for contact fields whose names being with 'Email':

```
https://secure.p03.eloqua.com/API/Bulk/2.0/contacts/fields?q=name='Email*'
```

**Important:** Search terms are *case sensitive*: if you search for 'email', you will not find any fields, as Eloqua's field names are capitalized.

The results should resemble:


```
{
   "count": 3,
   "hasMore": false,
   "items": [
        {
            "createdAt": "1900-01-01T05:00:00.0000000Z",
            "dataType": "emailAddress",
            "hasNotNullConstraint": false,
            "hasReadOnlyConstraint": false,
            "hasUniquenessConstraint": true,
            "internalName": "C_EmailAddress",
            "name": "Email Address",
            "statement": "{{Contact.Field(C_EmailAddress)}}",
            "updatedAt": "1900-01-01T05:00:00.0000000Z",
            "uri": "/contacts/fields/100001"
        },
        {
            "createdAt": "1900-01-01T05:00:00.0000000Z",
            "dataType": "string",
            "hasNotNullConstraint": false,
            "hasReadOnlyConstraint": false,
            "hasUniquenessConstraint": false,
            "internalName": "C_EmailDisplayName",
            "name": "Email Display Name",
            "statement": "{{Contact.Field(C_EmailDisplayName)}}",
            "updatedAt": "1900-01-01T05:00:00.0000000Z",
            "uri": "/contacts/fields/100005"
        },
        {
            "createdAt": "1900-01-01T05:00:00.0000000Z",
            "dataType": "string",
            "hasNotNullConstraint": false,
            "hasReadOnlyConstraint": false,
            "hasUniquenessConstraint": false,
            "internalName": "C_SFDC_EmailOptOut1",
            "name": "SFDC Email Opt-Out",
            "statement": "{{Contact.Field(C_SFDC_EmailOptOut1)}}",
            "updatedAt": "1900-01-01T05:00:00.0000000Z",
            "uri": "/contacts/fields/100043"
        }
   ]
}
```


The `statement` field shows the *Eloqua Markup Language* (page 29) representation of that field, which you can then use in an import or export definition.

## 2.4 Retrieve Entities from Eloqua with `GET`

You can execute HTTP `GET` to retrieve account, activity, contact, email group, and custom object entities from Eloqua, as well as import definitions, export definitions, logs, and syncs.

### 2.4.1 Basic Request

The basic `GET` request simply specifies the `id` of the entity that you wish to retrieve from Eloqua in the request. For example, the following call requests the contact import with `id` of 1:

```
GET https://<host>.eloqua.com/api/bulk/2.0/contacts/imports/1
```

The response might resemble:

```
{
    "createdAt": "2014-05-13T18:25:32.0270000Z",
    "createdBy": "Docs.Example",
    "fields": {
        "C_EmailAddress": "{{Contact.Field(C_EmailAddress)}}",
        "C_FirstName": "{{Contact.Field(C_FirstName)}}",
        "C_LastName": "{{Contact.Field(C_LastName)}}"
    },
    "identifierFieldName": "C_EmailAddress",
    "isSyncTriggeredOnImport": false,
    "isUpdatingMultipleMatchedRecords": false,
    "name": "Docs Import Example",
    "syncActions": [],
    "updatedAt": "2014-05-13T18:25:32.0270000Z",
    "updatedBy": "Docs.Example",
    "uri": "/contacts/imports/1"
}
```

---

**Note:** Many Eloqua's Bulk API `GET` endpoints include optional parameters that you can use to sort, order, and filter the results of the `GET` request. See: *URL Parameters* (page 3) for more information.

---

### 2.4.2 Retrieve a List of Entities

In some cases, you may already know the `id` of the element you wish to retrieve from Eloqua, but if you do not, you can retrieve a list to choose from.

1. Obtain the list of entities available at your endpoint. For example, if you wanted to retrieve a contact import, but did not know its `id`, you would call:

   ```
   GET https://<host>.eloqua.com/api/bulk/2.0/contacts/imports?
   ```

2. Select the appropriate element from the list. The response to the `/contacts/imports?` get request will resemble:

   ```
   {
      "items": [
         {
             "createdAt": "2013-04-17T16:40:43.0270000Z",
             "createdBy": "Docs.Example",
             "dataRetentionDuration": "PT1H",
             "fields": {
                 "C_EmailAddress": "{{Contact.Field(C_EmailAddress)}}",
                 "C_FirstName": "{{Contact.Field(C_FirstName)}}",
                 "C_LastName": "{{Contact.Field(C_LastName)}}"
             },
             "identifierFieldName": "C_EmailAddress",
             "isSyncTriggeredOnImport": true,
             "isUpdatingMultipleMatchedRecords": false,
   ```

```
            "name": "Example 1",
            "syncActions": [],
            "updateRule": "always",
            "updatedAt": "2013-04-17T16:40:43.0270000Z",
            "updatedBy": "Docs.Example",
            "uri": "/contacts/imports/1"
        },
        {
            "createdAt": "2014-04-11T18:43:37.4430000Z",
            "createdBy": "Docs.Example",
            "dataRetentionDuration": "PT1H",
            "fields": {
                "C_Lead_Status1": "{{Contact.Field(C_Lead_Status1)}}"
            },
            "identifierFieldName": "C_Lead_Status1",
            "isSyncTriggeredOnImport": true,
            "isUpdatingMultipleMatchedRecords": false,
            "name": "Example 2",
            "syncActions": [
                {
                    "action": "add",
                    "destination": "{{ContactList[123]}}"
                }
            ],
            "updateRule": "always",
            "updatedAt": "2014-04-11T18:43:37.4430000Z",
            "updatedBy": "Docs.Example",
            "uri": "/contacts/imports/2"
        }
    ]
}
```


Locate the `uri` of the element you wish to access.

3. Request the element:

```
GET https://<host>/api/bulk/2.0/contacts/imports/2
```

You can also filter lists based on query criteria using the `q` *url parameter* (page 3). For an example, see: *Search for Field Names* (page 15).

## 2.5 Create a New Entity with `POST`

HTTP `POST` requests enable you to create a new entity within Eloqua. Essentially, the Bulk API's `POST` endpoints create imports and exports and upload data for import, and trigger the `sync` that actually imports data into Eloqua, or exports data out of Eloqua.

The following Eloqua's Bulk API endpoints support `POST` requests:

- **Accounts:**
  - `/accounts/exports`
  - `/accounts/imports`
  - `/accounts/imports/{id}/data`
- **Activities:**

  - /activities/exports
  - /activities/imports
  - /activities/imports/{id}/data
- **Contacts:**
  - /contacts/exports
  - /contacts/imports
  - /contacts/imports/{id}/data
- **Custom Objects:**
  - /customObjects/{parentId}/exports
  - /customObjects/{parentId}/imports
  - /customObjects/{parentId}/imports/{id}/data
- **Syncs:**
  - /syncs

For detailed instructions on importing and exporting data with Eloqua see: *Import Data Into Eloqua* (page 9) and *Export Data from Eloqua* (page 12).

### 2.5.1 Create a Contacts Import Entity

To create a contact import entity, simply POST to the /contacts/imports endpoints, as in the following example:

```
POST https://<host>.eloqua.com/API/Bulk/2.0/contacts/imports/
```

```
{
   "fields" : {
      "firstName" : "{{Contact.Field(C_FirstName)}}",
      "lastName" : "{{Contact.Field(C_LastName)}}",
      "email" : "{{Contact.Field(C_EmailAddress)}}"
   },
   "identifierFieldName" : "email",
   "name" : "Docs Example Contact Import"
}
```

The successful response returns the import:

```
{
    "createdAt": "2014-04-22T18:58:16.5500000Z",
    "createdBy": "Docs.Example",
    "fields": {
        "email": "{{Contact.Field(C_EmailAddress)}}",
        "firstName": "{{Contact.Field(C_FirstName)}}",
        "lastName": "{{Contact.Field(C_LastName)}}"
    },
    "identifierFieldName": "email",
    "isSyncTriggeredOnImport": false,
    "isUpdatingMultipleMatchedRecords": false,
    "name": "Docs Example Contact Import - April 2014",
    "syncActions": [],
    "updatedAt": "2014-04-22T18:58:16.5500000Z",
    "updatedBy": "Docs.Example",
```

```
    "uri": "/contacts/imports/1056"
}
```

A `POST` request to `/contacts/imports/1056/data` endpoint is then used to send the data to Eloqua.

## 2.6 Update an Existing Import or Export with `PUT`

HTTP `PUT` requests provide the ability to update existing imports and exports using Eloqua's Bulk API. HTTP `PUT` requests are available for the following endpoints:

- `/accounts/exports/{id}`
- `/accounts/imports/{id}`
- `/activities/imports/{id}`
- `/activities/exports/{id}`
- `/contacts/exports/{id}`
- `/contacts/imports/{id}`
- `/customObjects/{parentId}/exports/{id}`
- `/customObjects/{parentId}/imports/{id}`

### 2.6.1 Update Account Import Definition with `PUT`

Consider the following account import:

```
{
   "createdAt": "2014-01-07T16:03:08.8030000Z",
   "createdBy": "Docs.Example",
   "dataRetentionDuration": "PT1H",
   "fields": {
       "CompanyName": "{{Account.Field(M_CompanyName)}}"
   },
   "identifierFieldName": "CompanyName",
   "isSyncTriggeredOnImport": true,
   "isUpdatingMultipleMatchedRecords": false,
   "kbUsed": 0,
   "name": "This Import Name is Not Helpful",
   "syncActions": [],
   "updateRule": "always",
   "updatedAt": "2014-01-07T16:03:08.8030000Z",
   "updatedBy": "Docs.Example",
   "uri": "/accounts/imports/892"
}
```

The `name`, "This Import Name is Not Helpful" is not as descriptive as it should be to be useful in the future. So, we can use a `PUT` request to update it to "Docs Account Import Example":

```
PUT https://<host>.eloqua.com/API/Bulk/2.0/accounts/imports/892
```

```
{
   "fields" : {
       "CompanyName": "{{Account.Field(M_CompanyName)}}"
   },
```

```
    "identifierFieldName": "CompanyName",
    "name" : "Docs Example Account Import"
}
```

The `identifierFieldName` and `fields` fields are required for the call to be successful. Since we wanted to update the `name` field, that is also included.

When we `GET` the import info, the response reflects the updated name: `GET https://<host>.eloqua.com/API/Bulk/2.0/accounts/imports/892` returns:

```
{
    "createdAt": "2014-01-07T16:03:08.8030000Z",
    "createdBy": "Docs.Example",
    "fields": {
        "CompanyName": "{{Account.Field(M_CompanyName)}}"
    },
    "identifierFieldName": "M_CompanyName",
    "isSyncTriggeredOnImport": false,
    "isUpdatingMultipleMatchedRecords": false,
    "kbUsed": 0,
    "name": "Docs Example Account Import",
    "syncActions": [],
    "updatedAt": "2014-04-22T17:56:34.2530000Z",
    "updatedBy": "Docs.Example",
    "uri": "/accounts/imports/892"
}
```

To add fields to the import, you follow the same pattern, simply adding additional fields in the `fields` parameter, and leaving out the `name` update:

```
PUT https://<host>.eloqua.com/API/Bulk/2.0/accounts/imports/892
```

```
{
    "fields": {
        "CompanyName": "{{Account.Field(M_CompanyName)}}",
        "Country": "{{Account.Field(M_Country)}}",
        "Website": "{{Account.Field(M_Website1)}}"
    },
    "identifierFieldName": "CompanyName"
}
```

## 2.7 Delete an Entity with `DELETE`

You can delete import and export entities from Eloqua using an HTTP `DELETE` request. The following endpoints support `DELETE` requests:

- `/accounts/exports/{id}`
- `/accounts/exports/{id}/data`
- `/accounts/imports/{id}`
- `/accounts/imports/{id}/data`
- `/activities/exports/{id}`
- `/activities/exports/{id}/data`
- `/activities/imports/{id}`

- `/activities/imports/{id}/data`
- `/contacts/exports/{id}`
- `/contacts/exports/{id}/data`
- `/contacts/imports/{id}`
- `/contacts/imports/{id}/data`
- `/customObjects/{parentId}/exports/{id}`
- `/customObjects/{parentId}/exports/{id}/data`
- `/customObjects/{parentId}/imports/{id}`
- `/customObjects/{parentId}/imports/{id}/data`

To request that an entity be delete, create an HTTP `DELETE` request, specifying the entity you wish to delete using its `{id}` as a URL parameter. The following example requests the deletion of the data from the contacts export with `id` 7:

```
DELETE https://<host>.eloqua.com/api/bulk/2.0/contacts/exports/7/data
```

If the request is successful, Eloqua returns a `204` response. Refer to *the status codes reference* (page 43) for other possible responses.

## 2.8 Authenticate Using OAuth

OAuth 2.0 authorization enables applications to act on behalf of another user without having to have that user's username and password credentials.

As an App Provider, you can use OAuth at any point that you need it: some Apps may need to authenticate during the configuration phase when a user installs the App in an instance of Eloqua; others may need OAuth when a user invokes a service. You can request as many access tokens as you need.

### 2.8.1 Introduction

The OAuth 2.0 Spec describes the original vision for OAuth2, which includes four possible flows that an application can use to obtain access on behalf of a resource owner: Authorization Code grant, Implicit grant, Resource Owner Password Credentials grant, and Client Credentials grant.

Eloqua's implementation of OAuth supports all of the grant types **except** Client Credentials grant.

#### General Flow

In general, OAuth authentication follows a six step pattern:

1. An application requires access to a protected resource and requests authorization from the resource owner.

   For example, Sally has installed an App in her instance of Eloqua that needs to interact with Eloqua's Bulk API on Sally's behalf. The App requests access to act on Sally's behalf.

2. The application obtains an Authorization Grant, either though the Authorization Code grant, Implicit grant, or Resource Owner Password Credentials grant.

3. The client requests an Access Token by authenticating with the authorization server using the authorization grant.

4. The authorization server validates the grant and issues an Access Token and a Refresh Token.
5. The client requests the protected resource, authenticating using the Access Token.
6. The resource server verifies the Access Token and if so, serves the request.

#### Endpoints

Authorization endpoint: `https://login.eloqua.com/auth/oauth2/authorize` Token endpoint: `https://login.eloqua.com/auth/oauth2/token`

### 2.8.2 Sample Flows with Eloqua Endpoints

There are three general flows that you can use to authenticate with OAuth. Each flow, or grant type, has its own strengths. In general, you should use the Authorization Code grant for Apps that extend Eloqua's functionality.

Eloqua also supports Implicit grant and Resource Owner Password Credential grant for use cases where those are more appropriate.

---

**Note:** The following examples assume that all requests are successful. For a detailed description of possible request responses (including a variety of failure types), see: *OAuth Reference* (page 46).

---

#### Authorization Code Grant

1. The Application needs to interact with Eloqua on Sally's behalf. The App directs Sally to `login.eloqua.com` at the `/auth/oauth2/authorize` endpoint.

   `/auth/oauth2/authorize` has five possible URL parameters:

| Parameter | Value | Required? |
|---|---|---|
| `response_type` | Must be `code` | Yes |
| `client_id` | Your App's client identifier | Yes |
| `redirect_uri` | Your App's registered redirection endpoint | Yes [1] |
| `scope` | Must be "full" or not supplied | No |
| `state` | An optional value that has meaning for your App | No |

   So, for example, the URL that the App directs Sally to might resemble:

```
https://login.eloqua.com/auth/oauth2/authorize?response_type=code
&client_id=a1b2c3d4&redirect_uri=https://client.example.com/cb
&scope=full&state=xyz
```

2. Sally accepts your App's request to access Eloqua on her behalf. She is then redirected to your app's redirection endpoint, the `redirect_url`, with an authorization code in the `code` URL parameter, as in the following example authorization dialog:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?code=SplxlOBeZQQYbYS6WxSbIA&state=xyz
```

3. Your App can now use the Authentication Code to obtain *Access and Refresh Tokens* using a `POST` request to the `login.eloqua.com/auth/oauth2/token` endpoint. The `POST` request should include a **JSON body** with the following parameters:

[1] We require all apps to register their redirection endpoint.

| Parameter | Value | Required? |
|---|---|---|
| grant_type | Must be authorization_code | Yes |
| code | The authorization code | Yes |
| redirect_uri | Your App's registered redirection endpoint | Yes |

The request must also authenticate your app using HTTP basic authentication using your with your App's client identifier as the username and your App's client secret as the password. The system **does not** support passing 'client_id' or client_secret parameters in the JSON body. You **must** use HTTP basic authentication.

The following call requests an Access Token using the authorization code obtained previously.

```
POST https://login.eloqua.com/auth/oauth2/token
Authorization: Basic Q09NUEFOWVhcdXNlcjE6cGFzc3dvcmQxMjM=

{
   "grant_type":"authorization_code",
   "code":"SplxlOBeZQQYbYS6WxSbIA",
   "redirect_uri":"https://client.example.com/cb"
}
```

4. The authorization server validates the authorization code and, if valid, responds with a JSON body containing the access token, refresh token, access token expiration time, and token type, as in the following:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "access_token":"2YotnFZFEjr1zCsicMWpAA",
   "token_type":"bearer",
   "expires_in":3600,
   "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA"
}
```

5. When the App needs a protected resource, it authenticates during the request with the access token.

   The following call to Eloqua's Rest API uses the access token to authenticate:

```
GET /resource/1 HTTP/1.1
Host: api.eloqua.com
Authorization: Bearer 2YotnFZFEjr1zCsicMWpAA
```

6. api.eloqua.com verifies the access token, and supplies the requested resource if the access token is valid. If the access token has expired, the App can send the *refresh token* to login.eloqua.co/auth/oauth2/token to obtain a new access token, as in the following example:

```
POST https://login.eloqua.com/auth/oauth2/token
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3

{
   "grant_type":"refresh_token",
   "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
   "scope":"full",
   "redirect_uri":"https://client.example.com/cb"
}
```

   If the request is successful, the response is a JSON body containing a new access token, refresh token, access token expiration time and token type:

   HTTP/1.1 200 OK Content-Type: application/json

```
{
   "access_token":"2YotnFZFEjr1zCsicMWpAA",
   "token_type":"bearer",
   "expires_in":3600,
   "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA"
}
```

And then proceed to request the resource again using the new access token.

### Implicit Grant

Obtaining an access token using implicit grant varies slightly from the general flow. Implicit grant is a browser-based authentication, and instead of issuing an authorization code that is exchanged for an access token, the App is issued an access token directly.

This authentication approach is best used for in-browser applications, but authorization code grant is preferred and has fewer potentially negative security implications.

1. The Application needs to interact with Eloqua on Sally's behalf. To obtain an access token, the App initiates the authorization process through a `GET` request to `login.eloqua.com/auth/oauth2/authorize` with the following URL parameters:

| Parameter | Value | Required? |
|---|---|---|
| `response_type` | Must be `token` | Yes |
| `client_id` | Your App's client identifier | Yes |
| `redirect_uri` | Your App's registered redirection endpoint | Yes |
| `scope` | Must be "full" or not supplied | No |
| `state` | An optional value that has meaning for your App | No |

   For example:

   ```
   https://login.eloqua.com/auth/oauth2/authorize?response_type=token
   &client_id=s6BhdRkqt3&redirect_uri=https://client.example.com/app
   &scope=full&state=xyz
   ```

2. Sally accepts the App's request to access Eloqua on her behalf. She is then redirected to the App's redirection endpoint, the `redirect_url`, with an authorization code in the `access_token` fragment parameter, as in the following example authorization dialog:

   ```
   HTTP/1.1 302 Found
   Location: https://client.example.com/cb#access_token=2YotnFZFEjr1zCsicMWpAA
   &token_type=bearer&expires_in=86400&state=xyz
   ```

The App can then use this access token to request resources on Sally's behalf.

### Resource Owner Password Credentials Grant

With the resource owner password credential grant, the App provides the resource owner's username and password to the authorization server in order to obtain an access token. This grant type is ideal when an App already has usernames and passwords, and wants to start using access tokens instead.

To obtain access and refresh tokens, your app must `POST` to `/auth/oauth2/token` with a JSON body containing the following parameters:

| Parameter | Value | Required? |
|---|---|---|
| `grant_type` | Must be `password` | Yes |
| `scope` | Must be `full` or not supplied | No |
| `username` | The user's site name and username in the form `sitename` + '/' + `username` | Yes |
| `password` | The user's password | Yes |

The request must also authenticate your app using HTTP basic authentication using your with your App's client identifier as the username and your App's client secret as the password. The system **does not** support passing `'client_id'` or `client_secret` parameters in the JSON body. You **must** use HTTP basic authentication.

The following call requests an Access Token using the username and password.

```
POST https://login.eloqua.com/auth/oauth2/token
Authorization: Basic Q09NUEFOWVhcdXNlcjE6cGFzc3dvcmQxMjM=

{
   "grant_type":"password",
   "scope":"full",
   "username":"testsite\\sally",
   "password":"sally123"
}
```

If the request is successful, the response is a JSON body containing the access token, refresh token, access token expiration time, and token type:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "access_token":"2YotnFZFEjr1zCsicMWpAA",
   "token_type":"bearer",
   "expires_in":3600,
   "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA"
}
```

## 2.9 Authenticate using HTTP Basic Authentication

For HTTP basic authentication, each request must include an Authentication header, with a base-64 encoded value.

Your authentication token is of the format:

```
siteName + '\' + username + ':' + password
```

Where `siteName` is the company name you use to log in to Eloqua, and `username` and `password` are your Eloqua username and password.

For example, for a user whose company name is "COMPANYX", username is "user1", and password is "password123", the base string is:

```
COMPANYX\user1:password123
```

Running base-64 encoding on that string gives the token for the Authentication header:

```
Q09NUEFOWVhcdXNlcjE6cGFzc3dvcmQxMjM=
```

The authentication header would then be:

```
Authorization: Basic Q09NUEFOWVhcdXNlcjE6cGFzc3dvcmQxMjM=
```

CHAPTER

THREE

# REFERENCE

## 3.1 Eloqua Markup Language version 3

**Important:** This document discusses Eloqua Markup Language v3. If you are looking for documentation of Eloqua Markup Language v2, see: TopLiners.

Eloqua Markup Language (EML) is a text-based language you can use to access data in the Eloqua system that provides a consistent view of the data you care about.

EML uses text to identify fields and data relationships for data-centric entities. This way, you can easily describe and access the field values, entity attributes and relations that are important to you, using intuitive language.

### 3.1.1 Available Entities

EML allows you to access Contact, Account, and Custom Object data within your Eloqua database. You can also access related assets, providing the relationship resolves to a single entity.

### 3.1.2 Access Patterns

The general access format is `{{EntityName.InfoYouWant}})`, where `InfoYouWant` is a field name or attribute. The syntax varies slightly between those two types of data: for *named fields*, the syntax is `{{EntityName.Field(FieldName)}}`, while for *attributes*, the syntax is `{{EntityName.AttributeName}}`.

EML also uses dot notation to connect related assets, in the form `{{EntityName.RelatedAssetName.InfoYouWant}}`.

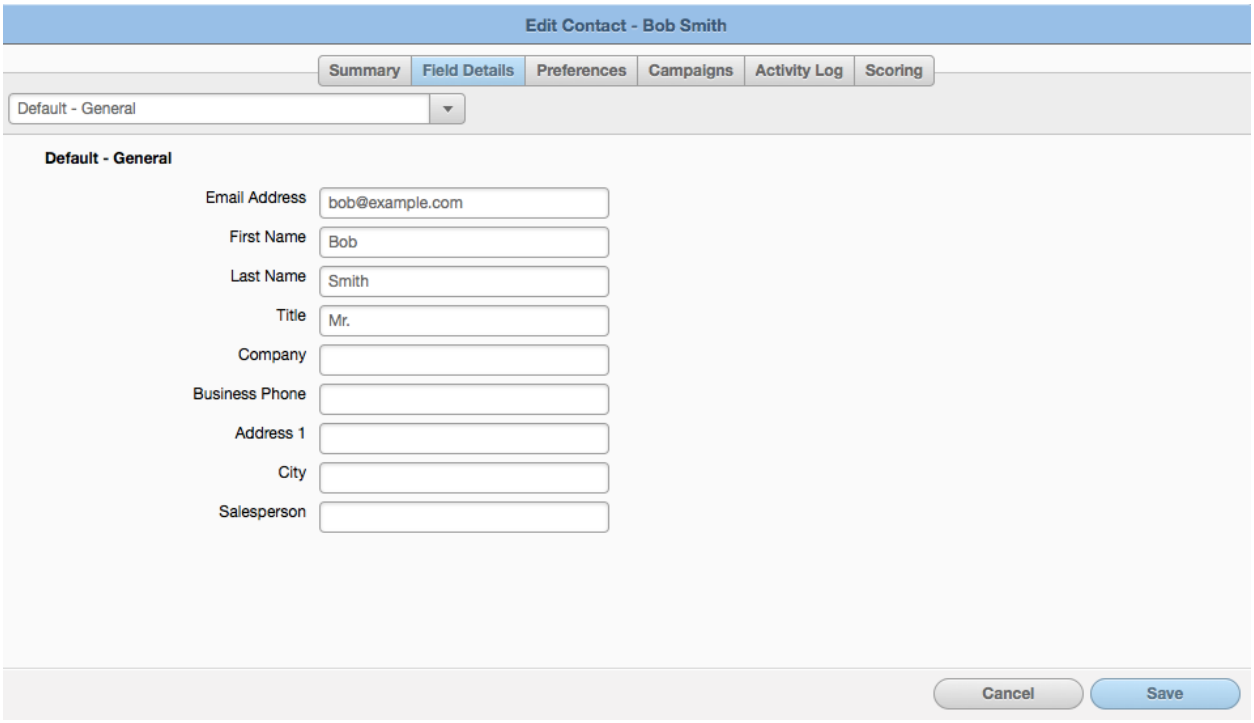
There are some syntax requirements to bear in mind:

- The system resolves all statements enclosed in `{{` and `}}` if it can parse them. You must put statements that require resolution between double braces.
- To access *attributes*, use brackets `[ ]` to delimit the requested attribute. To access *fields*, use parentheses `( )` to delimit the named field.
- When specifying a field or asset using ELM, you must refer to the *internal name*, rather than the *display name*. The display name is the name that appears in the Eloqua UI, while the internal name is the identifier used by the system. The Contact Entity's email address field, for example, has the display name: "Email Address" and the internal name "C_EmailAddress". See: *Search for Field Names* (page 15) for more information.

You can only access a related asset that resolves to a single record. For instance, since a Contact is associated with at most one Account, you can retrieve a Contact's related Account information. Conversely, since Accounts are linked to many Contacts, you cannot retrieve related Contact information for an Account.

### 3.1.3 Syntax Examples

These examples use a fictional contact called Bob Smith throughout. Bob's contact information resembles the following:




#### Static Text

Static text is the most straightforward scenario: the EML definition `This is static text` simply resolves to *This is static text*.

#### Field Values

To access field values, the format is `{{Type.Field(fieldName)}}` For instance, to access the `C_FirstName` field in a Contact, the markup would resemble:

```
{{Contact.Field(C_FirstName)}}
```

For contact Bob Smith, the definition resolves to *Bob*.

#### Entity Attributes

You can also access an entity's attributes. The definition is extremely similar to the markup for accessing entity fields. To access a contact's `Id`, the internal contact identifier that Eloqua generates and stores when you add a new contact, you would input:

```
{{Contact.Id}}
```

For Bob Smith, whose internal contact identifier is 123456, `{{Contact.Id}}` resolves to *123456*.

#### Related Entities

To access a Contact's linked Account information, in this case, `M_CompanyName`, the definition is:

```
{{Contact.Account.Field(M_CompanyName)}}
```

For Bob Smith, who is a contact of Eloqua, this resolves to *Eloqua*.

#### Combinations of Values

EML definitions can include multiple field or attribute requests: the system returns the concatenated values.

For instance, if you wanted to create a full name, with their Account name, your definition would resemble:

```
{{Contact.Field(C_FirstName)}} {{Contact.Field(C_LastName)}}, of
{{Contact.Account.Field(M_CompanyName)}}
```

For Contact Bob Smith, this resolves to *Bob Smith, of Eloqua*.

### 3.1.4 Changes In Version 3

Eloqua Markup Language v3 builds on the previous version of EML to simplify and streamline the ways you describe and access data within Eloqua. EML v3 adds support for accessing visitor information for activities coming in from the web: for example, when someone opens an email sent through Eloqua. The changes are primarily structural, making it easier to access the data you want. To that end, v3 introduces three new tokens: `ASSET_TYPE_ATTRIBUTE`, `VISITOR_ID_ATTRIBUTE`, and `VISITOR_EXTERNAL_ID_ATTRIBUTE`.

### 3.1.5 Grammar

```
grammar Eml;

options {
    language = CSharp3;
    output = AST;
    backtrack = true;
}

tokens {
    FIELD_NAME;
    FIELD_INDEX;
    ACTIVITY_TYPE_ATTRIBUTE;
    ASSET_TYPE_ATTRIBUTE;
    ASSET_NAME_ATTRIBUTE;
    ASSET_ID_ATTRIBUTE;
    CAMPAIGN_ID_ATTRIBUTE;
    VISITOR_ID_ATTRIBUTE;
    VISITOR_EXTERNAL_ID_ATTRIBUTE;

}

@modifier{public}
@parser::namespace { Eloqua.Markup.Version3.Grammar }
@lexer::namespace { Eloqua.Markup.Version3.Grammar }

public dynamicEntity
```

```
    : DYNAMICSTART! (dataEntity | assetEntity) DYNAMICEND!
    ;

assetEntity
    : contactContainerAsset
    | emailGroupAsset
    | accountContainerAsset
    | cloudInstanceAsset
    ;

contactContainerAsset
    : CONTACTLIST INDEXED_ID
    | CONTACTFILTER INDEXED_ID
    | CONTACTSEGMENT INDEXED_ID
    ;

emailGroupAsset
    : EMAILGROUP INDEXED_ID
    | EMAIL INDEXED_ID DOT! GROUP
    ;

accountContainerAsset
    : ACCOUNTLIST INDEXED_ID
    ;

cloudInstanceAsset
    : CONTENTINSTANCE NAMED_ID (DOT! execution)?
    | ACTIONINSTANCE NAMED_ID (DOT! execution)?
    | DECISIONINSTANCE NAMED_ID (DOT! execution)?
    | FEEDERINSTANCE NAMED_ID
    ;

dataEntity
    : relationshipToContact DOT! contactAttribute
    | relationshipToAccount DOT! accountAttribute
    | relationshipToCustomObject DOT! customObjectAttribute
    | relationshipToActivity DOT! activityAttribute
    ;

relationshipToContact
    : CONTACT
    | customObject DOT! CONTACT
    | ACTIVITY DOT! CONTACT
    ;

relationshipToAccount
    : ACCOUNT
    | CONTACT DOT! ACCOUNT
    | customObject DOT! ACCOUNT
    ;

relationshipToCustomObject
    : customObject
    ;

relationshipToActivity
    : ACTIVITY
    ;
```

```
contactAttribute
    : field
    | ID_ATTRIBUTE
    | CREATEDAT_ATTRIBUTE
    | UPDATEDAT_ATTRIBUTE
    | EMAIL! DOT! (
        | EMAIL_ISSUBSCRIBED_ATTRIBUTE
        | EMAIL_ISBOUNCED_ATTRIBUTE
        | EMAIL_FORMAT_ATTRIBUTE
    );

accountAttribute
    : field
    | ID_ATTRIBUTE
    | CREATEDAT_ATTRIBUTE
    | UPDATEDAT_ATTRIBUTE
    ;

customObjectAttribute
    : field
    | ID_ATTRIBUTE
    | CREATEDAT_ATTRIBUTE
    | UPDATEDAT_ATTRIBUTE
    ;

activityAttribute
    : field
    | ID_ATTRIBUTE
    | TYPE_ATTRIBUTE -> ACTIVITY_TYPE_ATTRIBUTE
    | CREATEDAT_ATTRIBUTE
    | CAMPAIGN DOT ID_ATTRIBUTE -> CAMPAIGN_ID_ATTRIBUTE
    | VISITOR DOT (
        | ID_ATTRIBUTE -> VISITOR_ID_ATTRIBUTE
        | EXTERNAL_ID_ATTRIBUTE -> VISITOR_EXTERNAL_ID_ATTRIBUTE
    )
    | ASSET DOT (
        | TYPE_ATTRIBUTE -> ASSET_TYPE_ATTRIBUTE
        | NAME_ATTRIBUTE -> ASSET_NAME_ATTRIBUTE
        | ID_ATTRIBUTE -> ASSET_ID_ATTRIBUTE
    );

execution
    : EXECUTION INDEXED_ID
    ;

field
    : FIELD (
          NAMED_ID -> FIELD_NAME NAMED_ID
        | INDEXED_ID -> FIELD_INDEX INDEXED_ID
    );

customObject
    : CUSTOMOBJECT INDEXED_ID
    ;

// Lexer tokens
DYNAMICSTART : '{{' ;
DYNAMICEND   : '}}' ;
```

```
DOT          : '.'  ;

// roots
CONTACT                      : 'Contact';
ACCOUNT                      : 'Account';
CUSTOMOBJECT                 : 'CustomObject';
ACTIVITY                     : 'Activity';
CONTACTLIST                  : 'ContactList';
CONTACTFILTER                : 'ContactFilter';
CONTACTSEGMENT               : 'ContactSegment';
ACCOUNTLIST                  : 'AccountList';
EMAIL                        : 'Email';
EMAILGROUP                   : 'EmailGroup';
CONTENTINSTANCE              : 'ContentInstance';
ACTIONINSTANCE               : 'ActionInstance';
DECISIONINSTANCE             : 'DecisionInstance';
FEEDERINSTANCE               : 'FeederInstance';

// attributes
GROUP                        : 'Group';
CAMPAIGN                     : 'Campaign';
FIELD                        : 'Field';
ASSET                        : 'Asset';
VISITOR                      : 'Visitor';
ID_ATTRIBUTE                 : 'Id';
EXTERNAL_ID_ATTRIBUTE        : 'ExternalId';
EMAIL_ISSUBSCRIBED_ATTRIBUTE : 'IsSubscribed';
EMAIL_ISBOUNCED_ATTRIBUTE    : 'IsBounced';
EMAIL_FORMAT_ATTRIBUTE       : 'Format';
TYPE_ATTRIBUTE               : 'Type';
NAME_ATTRIBUTE               : 'Name';
CREATEDAT_ATTRIBUTE          : 'CreatedAt';
UPDATEDAT_ATTRIBUTE          : 'UpdatedAt';
EXECUTION                    : 'Execution';

NAMED_ID : '(' ('a'..'z'|'A'..'Z'|'0'..'9'|'_'|' ')+ ')';
INDEXED_ID : '[' ('1'..'9') ('0'..'9')* ']';
```

## 3.2 Eloqua Expression Language

The Eloqua Expression Language (EEL) support for incorporating logic into decision making and filtering within Eloqua. replaces the `ExportFilter` data transfer object that was present in Bulk 1.0. EEL simplifies and formalizes the syntax, providing a consistent way to interact with your data.

### 3.2.1 Operators

#### Comparison Operators

- >
- >=
- <
- <=

- `!=`
- `=`
- `~`

#### Logical Operators

- `AND`
- `OR`
- `NOT`

#### Existence Operators

- `EXISTS`: analogous to an `IN` operator, determines if the object or entity exists within the container
- `STATUS`: e.g. "subscribed" or "unsubscribed" for email; "yes", "no", or "pending" for an AppCloud Decision

### 3.2.2 Examples

**Note:** All operands must be wrapped in single quotes.

#### Comparison

Select contacts whose `CreatedAt` field is greater that 2013-12-31:

```
"filter" : "'{{Contact.CreatedAt}}' > '2013-12-31'"
```

Select contacts whose Account's Company Name is not Eloqua:

```
"filter" : "'{{Contact.Account.Field(M_CompanyName)}}' != 'Eloqua'"
```

#### Logical

Select contacts whose `CreatedAt` field is greater that 2013-12-31 and whose Account's company name is not Eloqua:

```
"filter" : "'{{Contact.CreatedAt}}' > '2013-12-31' AND
'{{Contact.Account.Field(M_CompanyName)}}' != 'Eloqua'"
```

Select contacts whose `CreatedAt` field is greater than 2013-12-31 and whose Account's company name is not Eloqua and whose `C_Country` field is not Canada or United States:

```
"filter" : "('{{Contact.CreatedAt}}' > '2013-12-31'
         AND
         '{{Contact.Account.Field(M_CompanyName)}}' != 'Eloqua')
         AND NOT
         ('{{Contact.Field(C_Country)}}' = 'Canada'
         OR '{{Contact.Field(C_Country)}}' = 'United States')"
```

#### Existence

`EXISTS` refers specifically to a container. The following filters contacts based on their presence in the ContactList with `id` 123:

```
"filter" : "EXISTS('{{ContactList[123]}}')"
```

`STATUS`. The following filter selects contacts that are subscribed to the EmailGroup with id 123:

```
"filter" : "STATUS('{{EmailGroup[123]}}') = 'subscribed'"
```

### 3.2.3 Grammar

```
grammar Ebl;

options {
    language = CSharp3;
    output = AST;
    backtrack = true;
}

@modifier{public}
@parser::namespace { Eloqua.Expression.Version1.Grammar }
@lexer::namespace { Eloqua.Expression.Version1.Grammar }

public expression
    : WS!? orExpression WS!? EOF
    ;

orExpression
    : andExpression (WS!? OR^ WS!? andExpression)*
    ;

andExpression
    : notExpression (WS!? AND^ WS!? notExpression)*
    ;

notExpression
    : (NOT WS!?)? atom
    ;

subExpression
    : OPENPAREN! orExpression WS!? CLOSEPAREN!
    ;

atom
    : comparison
    | subExpression
    ;

comparison
    : STRING WS!? GREATERTHAN WS!? STRING
    | STRING WS!? GREATERTHANOREQUAL WS!? STRING
    | STRING WS!? LESSTHAN WS!? STRING
    | STRING WS!? LESSTHANOREQUAL WS!? STRING
    | STRING WS!? NOTEQUAL WS!? STRING
    | STRING WS!? EQUAL WS!? STRING
```

```
    | STRING WS!? LIKE WS!? STRING
    | EXISTS WS!? OPENPAREN! WS!? STRING WS!? CLOSEPAREN!
    | STATUS WS!? OPENPAREN! WS!? STRING WS!? CLOSEPAREN! WS!? EQUAL! WS!? STRING
    ;

// Lexer tokens

// Comparison Operators
GREATERTHAN         : '>'  ;
GREATERTHANOREQUAL  : '>=' ;
LESSTHAN            : '<'  ;
LESSTHANOREQUAL     : '<=' ;
NOTEQUAL            : '!=' ;
EQUAL               : '='  ;
LIKE                : '~'  ;

// Existence Operators
EXISTS              : E X I S T S ;
STATUS              : S T A T U S ;

// Binary
AND                 : A N D;
OR                  : O R;

// Unary
NOT                    : N O T;

// Grouping
OPENPAREN           : '('  ;
CLOSEPAREN          : ')'  ;

// Lexer rules
STRING    :  QUOTE ( ESC_SEQ | ~('\\'|'\'') )* QUOTE;
WS        : (' '|'\t'|'\n'|'\r')+ ;

fragment
ESC_SEQ
    :   '\\' ('b'|'t'|'n'|'f'|'r'|'\"'|'\''|'\\')
    |   '\\' 'u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT
    ;

fragment QUOTE     :  '\'' ;
fragment HEX_DIGIT : ('0'..'9'|'a'..'f'|'A'..'F') ;

fragment A:('a'|'A');
fragment B:('b'|'B');
fragment C:('c'|'C');
fragment D:('d'|'D');
fragment E:('e'|'E');
fragment F:('f'|'F');
fragment G:('g'|'G');
fragment H:('h'|'H');
fragment I:('i'|'I');
fragment J:('j'|'J');
fragment K:('k'|'K');
fragment L:('l'|'L');
fragment M:('m'|'M');
fragment N:('n'|'N');
```

```
fragment O:('o'|'O');
fragment P:('p'|'P');
fragment Q:('q'|'Q');
fragment R:('r'|'R');
fragment S:('s'|'S');
fragment T:('t'|'T');
fragment U:('u'|'U');
fragment V:('v'|'V');
fragment W:('w'|'W');
fragment X:('x'|'X');
fragment Y:('y'|'Y');
fragment Z:('z'|'Z');
```

## 3.3 Activity Fields

Eloqua's Bulk API supports five activity types. Each activity type has its own set of fields that you can interact with:

### 3.3.1 EmailOpen

```
{
   "ActivityId":"{{Activity.Id}}",
   "ActivityType":"{{Activity.Type}}",
   "ActivityDate":"{{Activity.CreatedAt}}",
   "EmailAddress":"{{Activity.Field(EmailAddress)}}",
   "ContactId":"{{Activity.Contact.Id}}",
   "IpAddress":"{{Activity.Field(IpAddress)}}",
   "VisitorId":"{{Activity.Visitor.Id}}",
   "EmailRecipientId":"{{Activity.Field(EmailRecipientId)}}",
   "AssetType":"{{Activity.Asset.Type}}",
   "AssetName":"{{Activity.Asset.Name}}",
   "AssetId":"{{Activity.Asset.Id}}",
   "SubjectLine":"{{Activity.Field(SubjectLine)}}",
   "EmailWebLink":"{{Activity.Field(EmailWebLink)}}"
}
```

### 3.3.2 EmailClickthrough

```
{
   "ActivityId":"{{Activity.Id}}",
   "ActivityType":"{{Activity.Type}}",
   "ActivityDate":"{{Activity.CreatedAt}}",
   "EmailAddress":"{{Activity.Field(EmailAddress)}}",
   "ContactId":"{{Activity.Contact.Id}}",
   "IpAddress":"{{Activity.Field(IpAddress)}}",
   "VisitorId":"{{Activity.Visitor.Id}}",
   "EmailRecipientId":"{{Activity.Field(EmailRecipientId)}}",
   "AssetType":"{{Activity.Asset.Type}}",
   "AssetName":"{{Activity.Asset.Name}}",
   "AssetId":"{{Activity.Asset.Id}}",
   "SubjectLine":"{{Activity.Field(SubjectLine)}}",
   "EmailWebLink":"{{Activity.Field(EmailWebLink)}}",
   "EmailClickthruLink":"{{Activity.Field(EmailClickthruLink)}}"
}
```

### 3.3.3 EmailSend

```
{
   "ActivityId":"{{Activity.Id}}",
   "ActivityType":"{{Activity.Type}}",
   "ActivityDate":"{{Activity.CreatedAt}}",
   "EmailAddress":"{{Activity.Field(EmailAddress)}}",
   "EmailRecipientId":"{{Activity.Field(EmailRecipientId)}}",
   "AssetType":"{{Activity.Asset.Type}}",
   "AssetId":"{{Activity.Asset.Id}}",
   "AssetName":"{{Activity.Asset.Name}}",
   "SubjectLine":"{{Activity.Field(SubjectLine)}}",
   "EmailWebLink":"{{Activity.Field(EmailWebLink)}}"
}
```

### 3.3.4 Unsubscribe

```
{
   "ActivityId":"{{Activity.Id}}",
   "ActivityType":"{{Activity.Type}}",
   "AssetId":"{{Activity.Asset.Id}}",
   "ActivityDate":"{{Activity.CreatedAt}}",
   "EmailAddress":"{{Activity.Field(EmailAddress)}}",
   "EmailRecipientId":"{{Activity.Field(EmailRecipientId)}}",
   "AssetType":"{{Activity.Asset.Type}}",
   "AssetName":"{{Activity.Asset.Name}}"
 }
```

### 3.3.5 Bounceback

```
{
   "ActivityId":"{{Activity.Id}}",
   "ActivityType":"{{Activity.Type}}",
   "ActivityDate":"{{Activity.CreatedAt}}",
   "EmailAddress":"{{Activity.Field(EmailAddress)}}"
}
```

## 3.4 Import Characteristics

Import characteristics are properties you use to control and describe an import with the Bulk API. They are **optional**

- `name` (**string**): name of this import (maximum 100 characters).
- `importPriorityUri` (**uri**): sets the priority of the data to import. It must reference an existing `/imports/priorities/{id}` URI. To see the list of the available priorities, send a `GET` request to `https://<host>.eloqua.com/api/bulk/2.0/imports/priorities?`.
- `isSyncTriggeredOnImport` (**boolean**): whether a sync is automatically created by the system during import.

  By default, `isSyncTriggeredOnImport` is `true`. If you set it to `false`, data that you push into the system will not be processed until you explicitly create a sync step for this import.

- `updateRule`: indicates whether to update values in the Eloqua database from values imported with this import definition. The available options are:
  - `Always`: Always update
  - `ifNewIsNotNull`: Update if the new value is not blank
  - `ifExistingIsNull`: Update if the existing value is not blank
  - `useFieldRule`: Use the rule defined at the field level
- `dataRetentionDuration` (**duration**): the length of time that unsync'd data from this import should remain in the staging area, in seconds. Valid values are anything from 3600 (1 hour) to 1209600 (2 weeks).
- `autoDeleteDuration` (**duration**): the length of time (in seconds) before the import definition will be automatically deleted [from somewhere]. If the property is not set, automatic deletion will not occur. This is typically used for *one-time-use* export definitions.
- `isUpdatingMultipleMatchedRecords` (**boolean**): when set to `true`, if Eloqua finds multiple matching records for the identifier field, Eloqua will update all of the fields that match. If set to `false`, Eloqua updates based on its internal algorithm.

## 3.5 Sync Actions

Sync Actions are parameters that you declare in an import or export definition that specify additional actions Eloqua should take when you import or export data.

### 3.5.1 Format

The `syncActions` field has three parameters: `action`, `destination`, and `status`.

- `action` specifies what action to perform: either `add`, `remove`, or `setStatus`.
- `destination` specifies where the action should take place. For `add` or `remove` actions, this would be the `uri` of a list or email group. For `setStatus` it might the `uri` of an AppCloud Action or Decision service.
- `status` specifies what the status should be set to for a `setStatus` action. The AppCloud Developer Framework uses these statuses extensively to determine what stages contacts are at in a campaign. There are 8 possible statuses:
  - `subscribed`
  - `unsubscribed`
  - `active`
  - `complete`
  - `pending`
  - `errored`
  - `yes`
  - `no`

You can specify multiple sync actions in a single import or export definition.

### 3.5.2 Example

The following import definition uses the `add` sync action to add the imported contacts to the contact list with ID 12345 and the `setStatus` sync action to set the status of the AppCloud decision with instance id 123 to "yes":

```
{
   "fields": {
      "C_EmailAddress": "{{Contact.Field(C_EmailAddress)}}",
      "C_FirstName": "{{Contact.Field(C_FirstName)}}",
      "C_LastName": "{{Contact.Field(C_LastName)}}"
   },
   "syncActions": [
      {
         "action": "add"
         "destinationUri": "/contact/list/12345",
      },
      {
         "action" : "setStatus",
         "destination" : "{{CloudDecision(123)}}",
         "status" : "yes"
      }
   ],
   "identifierFieldName": "C_EmailAddress"
}
```

## 3.6 Discovery

---

**Note:** The URL you need to call to access Eloqua's Bulk API depends on which "pod" your Eloqua instance is hosted on. The base url is: `https://<host>.eloqua.com/api/bulk/2.0`, where `<host>` can be `secure.p01`, `secure.p02`, or `secure.p03`. To find your URL, see: Determining Endpoint URLs on Topliners.

---

Eloqua uses Web Application Description Language (WADL) to describe the Bulk API. WADL is a machine-readable XML description commonly used to describe REST web services. The WADL description lists the resources that make up the Bulk API, and specifies how to interact with them.

### 3.6.1 Request URL

Eloqua's Bulk API's WADL description is located at: `https://<host>/api/bulk/2.0/wadl`. You do not need to be logged into your Eloqua instance to access it.

### 3.6.2 Response

The response lists the resources available in the Bulk API, their inputs, available methods, and possible request responses.

In addition, Eloqua's custom-defined status codes provide deeper insight into errors than HTTP response codes can. The *HTTP Status Codes* (page 43) reference page describes each code in detail.

The following code block shows a subset of the response to an `https://<host>/api/bulk/2.0/wadl` request. Specifically, it provides the definition for the `GetAccountExportSearch` method of the `/accounts/exports` resource:

```
<?xml version="1.0"?>
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:api="http://eloqua.com/bulk/2.0/schema"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://wadl.dev.java.net/2009/02">
  <grammars>
    <include href="wadl.xsd" />
  </grammars>
  <resources base="https://secure.eloqua.com/api/bulk/2.0">
    <resource path="accounts">
      <resource path="exports">
        <method id="GetAccountExportSearch" name="GET">
          <doc />
          <request>
            <param name="limit" type="api:positiveIntegerTo1000">
              <doc />
            </param>
            <param name="links" type="xsd:string">
              <doc />
            </param>
            <param name="offset" type="xsd:int">
              <doc />
            </param>
            <param name="orderBy" type="api:orderBy">
              <doc>
                <appInfo xmlns="">
                  <Constraints>
                    <Constraint>Available Ordering Terms: uri, id, name,
                    dataRetentionDuration, maxRecords, autoDeleteDuration,
                    kbUsed, createdBy, createdAt, updatedBy, updatedAt.</Constraint>
                  </Constraints>
                </appInfo>
              </doc>
            </param>
            <param name="q" type="xsd:string">
              <doc />
            </param>
            <param name="totalResults" type="xsd:boolean">
              <doc />
            </param>
          </request>
          <response status="200">
            <representation element="api:AccountExportSearchResponse"
            mediaType="application/json" />
          </response>
          <response status="301">
            <doc title="Login required." />
          </response>
          <response status="400">
            <doc title="There was a parsing error." />
            <representation element="api:FailureResponse" mediaType="application/json" />
          </response>
          <response status="400">
            <doc title="There was a missing reference." />
            <representation element="api:FailureResponse" mediaType="application/json" />
          </response>
          <response status="400">
            <doc title="There was a serialization error." />
            <representation element="api:FailureResponse" mediaType="application/json" />
```

```
    </response>
    <response status="400">
      <doc title="There was a validation error." />
      <representation element="api:FailureResponse" mediaType="application/json" />
    </response>
    <response status="401">
      <doc title="Login required." />
    </response>
    <response status="401">
      <doc title="You are not authorized to make this request." />
    </response>
    <response status="401">
      <doc title="Authentication error." />
    </response>
    <response status="403">
      <doc title="This service has not been enabled for your site." />
    </response>
    <response status="403">
      <doc title="XSRF Protection Failure." />
    </response>
    <response status="404">
      <doc title="The requested resource was not found." />
    </response>
    <response status="409">
      <doc title="There was a conflict." />
      <representation element="api:FailureResponse" mediaType="application/json" />
    </response>
    <response status="412">
      <doc title="The resource you are attempting to delete has dependencies,
      and cannot be deleted." />
    </response>
    <response status="413">
      <doc title="Storage space exceeded." />
    </response>
    <response status="500">
      <doc title="The service has encountered an error." />
    </response>
    <response status="503">
      <doc title="Service is unavailable." />
    </response>
    <response status="503">
      <doc title="There was a timeout processing the request." />
    </response>
  </method>
```

Each method has a `request` description that itemizes the parameters the request accepts and any constraints on those parameters, and `response` descriptions that lists each possible response type.

## 3.7 HTTP Status Codes

In order to provide more insight into errors and failures during API calls, Eloqua uses the following HTTP status codes.

**See also:**

*Eloqua Status Codes* (page 44) for Eloqua's custom status codes

- 200: “Success”
- 201: “Success”
- 204: “Success”
- 301: “Login required”
- 400: “There was a parsing error.”
- 400: “There was a missing reference.”
- 400: “There was a serialization error.”
- 400: “There was a validation error”
- 401: “Login required”
- 401: “You are not authorized to make this request”
- 403: “This service has not been enabled for your site.”
- 403: “XSRF Protection Failure”
- 404: “The requested resource was not found.”
- 409: “There was a conflict.”
- 412: “The resource you are attempting to delete has dependencies, and cannot be deleted”
- 413: “Storage space exceeded.”
- 500: “The service has encountered an error.”
- 503: “Service is unavailable.”
- 503: “There was a timeout processing the request”

## 3.8 Eloqua Status Codes

In addition to the standard HTTP status codes, Eloqua includes specific status codes to help inform you when performing operations with the Bulk API. Each status code has a number, as well as a human-readable message.

- `ELQ-00001`: Total records processed.
- `ELQ-00002`: Invalid email addresses.
- `ELQ-00003`: Total records remaining after duplicates are rejected.
- `ELQ-00004`: Contacts created.
- `ELQ-00005`: Accounts created.
- `ELQ-00007`: CustomObject data created.
- `ELQ-00011`: Duplicate email addresses.
- `ELQ-00012`: CustomObject data updated.
- `ELQ-00013`: Contacts deleted.
- `ELQ-00014`: Accounts deleted.
- `ELQ-00016`: Contacts email marked as bounced.
- `ELQ-00017`: Contacts email marked as unsubscribed.
- `ELQ-00022`: Contacts updated.

- ELQ-00023: Accounts updated.
- ELQ-00025: ustomObject data linked.
- ELQ-00026: Duplicate identifier.
- ELQ-00032: Malformed records.
- ELQ-00034: Total unmatched records.
- ELQ-00040: Invalid data.
- ELQ-00048: Contact already exists.
- ELQ-00049: Total new records.
- ELQ-00055: Campaign not found.
- ELQ-00059: Empty records.
- ELQ-00060: Total rejected records.
- ELQ-00061: Rejected records (missing fields).
- ELQ-00062: Duplicate account linkage.
- ELQ-00067: CustomObject data deleted.
- ELQ-00068: Assets created.
- ELQ-00069: Activities created.
- ELQ-00070: Activity Type not found.
- ELQ-00072: Asset Type not found.
- ELQ-00101: Sync processed for sync {syncId}, resulting in {status} status.
- ELQ-00102: Successfully exported members to csv file.
- ELQ-00103: Membership snapshot for export.
- ELQ-00104: Successfully converted file {file} ({kbUsed} kb) with {converter} converter.
- ELQ-00105: Error processing import staging file {file} on sync {syncId} for user {userId}.
- ELQ-00106: Successfully converted csv file to sqlite, taking {kbUsed} kb.
- ELQ-00107: There was an error processing the {type}.
- ELQ-00109: Unable to convert file {file} with {converter} converter. File will remain locked and will NOT be processed.
- ELQ-00110: Trying to unlock file {file} on import {importId} for user {userId}. File has been {lockedState}.
- ELQ-00111: Field ({field}) is not part of import definition, and will be ignored.
- ELQ-00112: `identifierFieldName` ({field}) must be contained within Fields, if specified.
- ELQ-00114: Added members to Account List ({listId}).
- ELQ-00115: Subscribed members to Email Group ({groupId}).
- ELQ-00116: Unsubscribed members from Email Group ({groupId}).
- ELQ-00117: Removed members from Account List ({listId}).
- ELQ-00118: Removed members from Contact List ({listId}).
- ELQ-00119: Added members to Contact List ({listId}).

- `ELQ-00120`: Updated members status to {status} in Cloud Connector Instance ({instanceId}).
- `ELQ-00121`: Field ({field}) is part of import definition, but not included in file {file}.
- `ELQ-00122`: Validation errors found on sync {syncId} for user {userId}. Sync will NOT proceed.
- `ELQ-00123`: identifierField will be ignored since Activities are never updated.
- `ELQ-00124`: There was no data available to process.
- `ELQ-00125`: {field} ({value}): {constraint}
- `ELQ-00126`: There are no fields to process in the file {file}.
- `ELQ-00127`: There are no mapped fields in the file {file}.

## 3.9 Glossary

**account** Name of a customer. Accounts are companies that have Eloqua subscriptions. [This definition needs improving].

**attribute** An attribute is data that is associated with a record, but which is *not* supplied by the user and does not have an identifier. Attributes include IDs, timestamps, etc.

**contact** Individuals with email addresses who exist in the Eloqua database. Eloqua maintains additional informatino on contacts, such as their names, phone numbers, company (account) details, etc.

**custom object** Custom objects store data that is related to contact or account records. These are custom to your Eloqua instance to support your specific use cases.

**external activity** External activities are user-uploaded data. External activities can include email opening, event data, and so on.

**field** A named attribute in Eloqua that has an identifier and an internal name. Fields contain information that users supply, such as email addresses, phone numbers, names, etc.

**special operator** Eloqua includes special fields to track email features. For example, `Email.IsSubscribed` provides global subscription information for that email address, while `Email.Format` records preferences for email formats.

**staging area** The area that you move data into when performing *imports* (page 9) or *exports* (page 12) with the Bulk API. The staging area system supports asynchronous processing of the imported and exported data, to mitigate potential performance issues. See: *the Pattern section* (page 2) of the API Call Format document.

## 3.10 OAuth Reference

### 3.10.1 Responses: Authorization Code Grant Request

#### Acceptance

If the user accepts your App's request to access Eloqua on their behalf, their user agent is eventually redirected to your app's redirection endpoint with an authorization code in the `code` URL parameter, as in the following example authorization dialog:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?code=SplxlOBeZQQYbYS6WxSbIA&state=xyz
```

### Rejection

If the user rejects your app's request to access Eloqua on their behalf, their user agent is eventually redirected to your App's registered redirection endpoint with the error `access_denied` in the `error` URL parameter, as in the following:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?error=access_denied&state=xyz
```

### Failure Before `client_id` or `redirect_url` Validation

If a failure occurs before the supplied client_id or redirect_uri are validated, we can't safely redirect the user agent back to the redirect URI to report the failure, and so we return the details of the failure in the body of the response.

#### Missing `client_id`

```
GET https://login.eloqua.com/auth/oauth2/authorize?response_type=code
&redirect_uri=https%3a%2f%2fclient.example.com%2fapp&scope=full&state=xyz

HTTP/1.1 200 OK
Content-Type: text/html

The &quot;client_id&quot; parameter is required.
```

#### Unknown `client_id`

```
GET https://login.eloqua.com/auth/oauth2/authorize?response_type=code
&client_id=00000000000000000000000000000000
&redirect_uri=https%3a%2f%2fclient.example.com%2fapp&scope=full&state=xyz

HTTP/1.1 200 OK
Content-Type: text/html

The &quot;client_id&quot; value is not a known client identifier.
```

#### Malformed `client_id`

```
GET https://login.eloqua.com/auth/oauth2/authorize?response_type=code
&client_id=malformed&redirect_uri=https%3a%2f%2fclient.example.com%2fapp
&scope=full&state=xyz

HTTP/1.1 200 OK
Content-Type: text/html

The &quot;client_id&quot; value is not a valid client identifier.
```

#### Missing `redirect_uri`

```
GET https://login.eloqua.com/auth/oauth2/authorize?response_type=code
&client_id=s6BhdRkqt3&scope=full&state=xyz
```

```
HTTP/1.1 200 OK
Content-Type: text/html

The &quot;redirect_uri&quot; parameter is required.
```

#### Malformed `redirect_uri`

```
GET https://login.eloqua.com/auth/oauth2/authorize?response_type=code
&client_id=s6BhdRkqt3&redirect_uri=malformed&scope=full&state=xyz

HTTP/1.1 200 OK
Content-Type: text/html

The &quot;redirect_uri&quot; value is not a valid URI.
```

#### Mismatched `redirect_uri`

```
GET https://login.eloqua.com/auth/oauth2/authorize?response_type=code
&client_id=s6BhdRkqt3&redirect_uri=https%3a%2f%2attacker.com%2fapp
&scope=full&state=xyz

HTTP/1.1 200 OK
Content-Type: text/html

The &quot;redirect_uri&quot; value doesn&#39;t start with the client redirect URI.
```

#### Non-HTTPS `redirect_uri`

```
GET https://login.eloqua.com/auth/oauth2/authorize?response_type=code
&client_id=s6BhdRkqt3&redirect_uri=http%3a%2f%2fclient.example.com%2fapp
&scope=full&state=xyz

HTTP/1.1 200 OK
Content-Type: text/html

The &quot;redirect_uri&quot; value is not an HTTPS URI.
```

#### `redirect_uri` with fragment

```
GET https://login.eloqua.com/auth/oauth2/authorize?response_type=code
&client_id=s6BhdRkqt3&redirect_uri=https%3a%2f%2fclient.example.com%2fapp%23fragment
&scope=full&state=xyz

HTTP/1.1 200 OK
Content-Type: text/html

The &quot;redirect_uri&quot; value has a fragment.
```

### Failure After `client_id` and `redirect_uri` Validation

If a failure occurs after the `client_id` and `redirect_uri` have been validated, Eloqua can safely redirect user agent back to the redirect URI to report the failure. In this case, the Authorization Dialog returns the details of the failure in the `error` and `error_description` URL parameters.

#### Internal server error

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?error=server_error
&error_description=The+server+encountered+an+unexpected+condition+that+prevented
+it+from+fulfilling+the+request.&state=xyz
```

#### Missing `response_type`

```
GET https://login.eloqua.com/auth/oauth2/authorize?
client_id=s6BhdRkqt3&redirect_uri=https%3a%2f%2fclient.example.com%2fapp
&scope=full&state=xyz

HTTP/1.1 302 Found
Location: https://client.example.com/cb?error=invalid_request
&error_description=The+%22response_type%22+parameter+is+required.&state=xyz
```

#### Unknown `response_type`

```
GET https://login.eloqua.com/auth/oauth2/authorize?response_type=unknown
&client_id=s6BhdRkqt3&redirect_uri=https%3a%2f%2fclient.example.com%2fapp
&scope=full&state=xyz

HTTP/1.1 302 Found
Location: https://client.example.com/cb?error=unsupported_response_type
&error_description=The+%22response_type%22+parameter+must+be+either+%22code%22
+or+%22token%22.&state=xyz
```

#### Unknown `scope`

```
GET https://login.eloqua.com/auth/oauth2/authorize?response_type=code
&client_id=s6BhdRkqt3&redirect_uri=https%3a%2f%2fclient.example.com%2fapp
&scope=unknown&state=xyz

HTTP/1.1 302 Found
Location: https://client.example.com/cb?error=invalid_scope
&error_description=The+%22scope%22+parameter+must+be+either+%22full%22+or
+not+supplied.&state=xyz
```

### 3.10.2 Responses: Exchanging Authorization Code for Access Token

#### Success

If the request is successful, the response will be a JSON body containing the access token, refresh token, access token expiration time, and token type:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "access_token":"2YotnFZFEjr1zCsicMWpAA",
   "token_type":"bearer",
   "expires_in":3600,
   "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA"
}
```

#### Failure

##### Missing, unknown, or malformed `client_id`, or missing or mismatched `client_secret`

We return the same response regardless of whether the `client_id` is missing, unknown, or malformed, or the `client_secret` is missing or mismatched.

Call:

```
POST https://login.eloqua.com/auth/oauth2/token
Authorization: Basic ...

{
   "grant_type":"authorization_code",
   "code":"SplxlOBeZQQYbYS6WxSbIA",
   "redirect_uri":"https://client.example.com/cb"
}
```

Yields:

```
HTTP/1.1 401 Unauthorized
Content-Type: application/json

{
   "error":"invalid_client",
   "error_description":"The client is invalid or was not supplied with basic authentication."
}
```

##### Missing code

Call:

```
POST https://login.eloqua.com/auth/oauth2/token
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3

{
   "grant_type":"authorization_code",
   "redirect_uri":"https://client.example.com/cb"
}
```

Yields:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
   "error":"invalid_request",
   "error_description":"The \"code\" parameter is required."
}
```

#### Unknown, malformed, expired, or invalidated code

Eloqua returns the same response regardless of whether the authorization code is unknown, malformed, expired, or invalid.

Call:

```
POST https://login.eloqua.com/auth/oauth2/token
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3

{
   "grant_type":"authorization_code",
   "code":"...",
   "redirect_uri":"https://client.example.com/cb"
}
```

Yields:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
   "error":"invalid_grant",
   "error_description":"The authorization code is incorrect, malformed, expired,
   or has been invalidated."
}
```

#### Missing `redirect_uri`

Call:

```
POST https://login.eloqua.com/auth/oauth2/token
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3

{
   "grant_type":"authorization_code",
   "code":"SplxlOBeZQQYbYS6WxSbIA"
}
```

Yields:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
   "error":"invalid_request",
```

```
   "error_description":"The \"redirect_uri\" parameter is required."
}
```

#### Malformed `redirect_uri`

Call:

```
POST https://login.eloqua.com/auth/oauth2/token
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3

{
   "grant_type":"authorization_code",
   "code":"SplxlOBeZQQYbYS6WxSbIA",
   "redirect_uri":"malformed"
}
```

Yields:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
   "error":"invalid_grant",
   "error_description":"The \"redirect_uri\" value is not a valid URI."
}
```

#### Mismatched `redirect_uri`

Call:

```
POST https://login.eloqua.com/auth/oauth2/token
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3

{
   "grant_type":"authorization_code",
   "code":"SplxlOBeZQQYbYS6WxSbIA",
   "redirect_uri":"https://attacker.com/cb"
}
```

Yields:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
   "error":"invalid_grant",
   "error_description":"The \"redirect_uri\" value doesn\u0027t start with the client
   redirect URI."
}
```

#### Non-HTTPS `redirect_uri`

Call:

```
POST https://login.eloqua.com/auth/oauth2/token
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3

{
   "grant_type":"authorization_code",
   "code":"SplxlOBeZQQYbYS6WxSbIA",
   "redirect_uri":"http://client.example.com/cb"
}
```

Yields:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
   "error":"invalid_grant",
   "error_description":"The \"redirect_uri\" value is not an HTTPS URI."
}
```

#### `redirect_uri` with fragment

Call:

```
POST https://login.eloqua.com/auth/oauth2/token
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3

{
   "grant_type":"authorization_code",
   "code":"SplxlOBeZQQYbYS6WxSbIA",
   "redirect_uri":"https://client.example.com/cb#fragment"
}
```

Yields:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
   "error":"invalid_grant",
   "error_description":"The \"redirect_uri\" value has a fragment."
}
```

### 3.10.3 Responses: Implicit Grant

#### Acceptance

If the user accepts your app's request to access Eloqua on their behalf, their user agent will eventually be redirected to your app's registered redirection endpoint with an authorization code in the `access_token` fragment parameter:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb#access_token=2YotnFZFEjr1zCsicMWpAA
&token_type=bearer&expires_in=86400&state=xyz
```

### Rejection

If the user rejects your app's request to access Eloqua on their behalf, their user agent will eventually be redirected to your app's registered redirection endpoint with the error `access_denied` in the `error` fragment parameter:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb#error=access_denied&state=xyz
```

### Failure before `client_id` or `redirect_uri` validation

If a failure occurs before the supplied `client_id` or `redirect_uri` are validated, we can't safely redirect the user agent back to the redirect URI to report the failure, and so we return the details of the failure in the body of the response.

#### Missing `client_id`

Call:

```
GET https://login.eloqua.com/auth/oauth2/authorize?response_type=token
&redirect_uri=https%3a%2f%2fclient.example.com%2fapp&scope=full&state=xyz
```

Yields:

```
HTTP/1.1 200 OK
Content-Type: text/html

The &quot;client_id&quot; parameter is required.
```

#### Unknown `client_id`

Call:

```
GET https://login.eloqua.com/auth/oauth2/authorize?response_type=token
&client_id=00000000000000000000000000000000
&redirect_uri=https%3a%2f%2fclient.example.com%2fapp&scope=full&state=xyz
```

Yields:

```
HTTP/1.1 200 OK
Content-Type: text/html

The &quot;client_id&quot; value is not a known client identifier.
```

#### Malformed `client_id`

Call:

```
GET https://login.eloqua.com/auth/oauth2/authorize?response_type=token
&client_id=malformed&redirect_uri=https%3a%2f%2fclient.example.com%2fapp
&scope=full&state=xyz
```

Yields:

```
HTTP/1.1 200 OK
Content-Type: text/html

The &quot;client_id&quot; value is not a valid client identifier.
```

#### Missing `redirect_uri`

Call:

```
GET https://login.eloqua.com/auth/oauth2/authorize?response_type=token
&client_id=s6BhdRkqt3&scope=full&state=xyz
```

Yields:

```
HTTP/1.1 200 OK
Content-Type: text/html

The &quot;redirect_uri&quot; parameter is required.
```

#### Malformed `redirect_uri`

Call:

```
GET https://login.eloqua.com/auth/oauth2/authorize?response_type=token
&client_id=s6BhdRkqt3&redirect_uri=malformed&scope=full&state=xyz
```

Yields:

```
HTTP/1.1 200 OK
Content-Type: text/html

The &quot;redirect_uri&quot; value is not a valid URI.
```

#### Mismatched `redirect_uri`

Call:

```
GET https://login.eloqua.com/auth/oauth2/authorize?response_type=token
&client_id=s6BhdRkqt3&redirect_uri=https%3a%2f%2attacker.com%2fapp
&scope=full&state=xyz
```

Yields:

```
HTTP/1.1 200 OK
Content-Type: text/html

The &quot;redirect_uri&quot; value doesn&#39;t start with the client redirect URI.
```

#### Non-HTTPS `redirect_uri`

Call:

```
GET https://login.eloqua.com/auth/oauth2/authorize?response_type=token
&client_id=s6BhdRkqt3&redirect_uri=http%3a%2f%2fclient.example.com%2fapp
&scope=full&state=xyz
```

Yields:

```
HTTP/1.1 200 OK
Content-Type: text/html

The &quot;redirect_uri&quot; value is not an HTTPS URI.
```

#### `redirect_uri` with fragment

Call:

```
GET https://login.eloqua.com/auth/oauth2/authorize?response_type=token
&client_id=s6BhdRkqt3&redirect_uri=https%3a%2f%2fclient.example.com%2fapp%23fragment
&scope=full&state=xyz
```

```
HTTP/1.1 200 OK
Content-Type: text/html

The &quot;redirect_uri&quot; value has a fragment.
```

### Failure After `client_id` and `redirect_uri` Validation

If a failure occurs after the supplied `client_id` and `redirect_uri` are validated, we can safely redirect the user agent back to the redirect URI to report the failure, and so we return the details of the failure in the `error` and `error_description` fragment parameters.

#### Internal server error

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb#error=server_error
&error_description=The+server+encountered+an+unexpected+condition
+that+prevented+it+from+fulfilling+the+request.&state=xyz
```

#### Missing `response_type`

If `response_type` is missing, we don't know whether to return the details of the failure in URL parameters or in fragment parameters. We've chosen to always return the details in URL parameters.

Call:

```
GET https://login.eloqua.com/auth/oauth2/authorize?client_id=s6BhdRkqt3
&redirect_uri=https%3a%2f%2fclient.example.com%2fapp&scope=full&state=xyz
```

Yields:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?error=invalid_request
&error_description=The+%22response_type%22+parameter+is+required.&state=xyz
```

#### Unknown `response_type`

If `response_type` is unknown, we don't know whether to return the details of the failure in URL parameters or in fragment parameters. We've chosen to always return the details in URL parameters.

Call:

```
GET https://login.eloqua.com/auth/oauth2/authorize?response_type=unknown
&client_id=s6BhdRkqt3&redirect_uri=https%3a%2f%2fclient.example.com%2fapp
&scope=full&state=xyz
```

Yields:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?error=unsupported_response_type
&error_description=The+%22response_type%22+parameter+must+be+either+%22code%22
+or+%22token%22.&state=xyz
```

#### Unknown `scope`

Call:

```
GET https://login.eloqua.com/auth/oauth2/authorize?response_type=token
&client_id=s6BhdRkqt3&redirect_uri=https%3a%2f%2fclient.example.com%2fapp
&scope=unknown&state=xyz
```

Yields:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb#error=invalid_scope
&error_description=The+%22scope%22+parameter+must+be+either+%22full%22
+or+not+supplied.&state=xyz
```

### 3.10.4 Response: Resource Owner Password Credentials Grant

#### Success

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "access_token":"2YotnFZFEjr1zCsicMWpAA",
   "token_type":"bearer",
   "expires_in":3600,
   "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA"
}
```

#### Failure

#### Missing, unknown, or malformed `client_id`, or missing or mismatched `client_secret`

We return the same response regardless of whether the `client_id` is missing, unknown, or malformed, or the `client_secret` is missing or mismatched.

Call:

```
POST https://login.eloqua.com/auth/oauth2/token
Authorization: Basic ...

{
   "grant_type":"password",
   "scope":"full",
   "username":"testsite\\testuser",
   "password":"Eloqua123"
}
```

Yields:

```
HTTP/1.1 401 Unauthorized

{
   "error":"invalid_client",
   "error_description":"The client is invalid or was not supplied with basic authentication."
}
```

#### Missing `username`

Call:

```
POST https://login.eloqua.com/auth/oauth2/token
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3

{
   "grant_type":"password",
   "scope":"full",
   "password":"Eloqua123"
}
```

Yields:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
   "error":"invalid_request",
   "error_description":"The \"username\" parameter is required."
}
```

#### Missing `password`

Call:

```
POST https://login.eloqua.com/auth/oauth2/token
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3

{
   "grant_type":"password",
   "scope":"full",
   "username":"testsite\\testuser"
}
```

Yields:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
   "error":"invalid_request",
   "error_description":"The \"password\" parameter is required."
}
```

#### Unknown site name, `username`, or `password`

We return the same response regardless of whether the site name, `username`, or `password` are unknown..

Call:

```
POST https://login.eloqua.com/auth/oauth2/token
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3

{
   "grant_type":"password",
   "scope":"full",
   "username":"...",
   "password":"..."
}
```

Yields:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
   "error":"invalid_grant",
   "error_description":"The site, username, or password are invalid."
}
```

#### Invalid `scope`

Call:

```
POST https://login.eloqua.com/auth/oauth2/token
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3

{
   "grant_type":"password",
   "scope":"invalid",
   "username":"testsite\\testuser",
   "password":"Eloqua123"
}
```

Yields:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
```

```
{
   "error":"invalid_scope",
   "error_description":"The \"scope\" parameter must be either \"full\" or not supplied."
}
```

### 3.10.5 Responses: Exchanging Refresh Token for New Access Token

#### Success

```
HTTP/1.1 200 OK
Content-Type: application/json

{
   "access_token":"2YotnFZFEjr1zCsicMWpAA",
   "token_type":"bearer",
   "expires_in":3600,
   "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA"
}
```

The refresh token returned may be different from the refresh token supplied. If this happens, the refresh token that was supplied will have been invalidated, and the refresh token returned must be used for all subsequent requests.

#### Failure

##### Missing, unknown, or malformed `client_id`, or missing or mismatched `client_secret`

We return the same response regardless of whether the `client_id` is missing, unknown, or malformed, or the `client_secret` is missing or mismatched.

Call:

```
Authorization: Basic ...

{
   "grant_type":"refresh_token",
   "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
   "scope":"full",
   "redirect_uri":"https://client.example.com/cb"
}
```

Yield response:

```
HTTP/1.1 401 Unauthorized
Content-Type: application/json

{
   "error":"invalid_client",
   "error_description":"The client is invalid or was not supplied with
   basic authentication."
}
```

#### Unknown, malformed, expired, or invalidated `refresh_token`

We return the same response regardless of whether the `refresh_token` is unknown, malformed, expired, or has been invalidated.

Call:

```
POST https://login.eloqua.com/auth/oauth2/token
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3

{
   "grant_type":"refresh_token",
   "refresh_token":"...",
   "scope":"full",
   "redirect_uri":"https://client.example.com/cb"
}
```

Yields response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
   "error":"invalid_grant",
   "error_description":"The refresh token is incorrect, malformed, expired,
   or has been invalidated."
}
```

#### Unknown `scope`

Call:

```
POST https://login.eloqua.com/auth/oauth2/token
Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbUl3

{
   "grant_type":"refresh_token",
   "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
   "scope":"unknown",
   "redirect_uri":"https://client.example.com/cb"
}
```

Yields response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
   "error":"invalid_scope",
   "error_description":"The \"scope\" parameter must be either \"full\" or not supplied."
}
```