ORACLE

Oracle Eloqua AppCloud Developer Framework

Developer Guide

27-Apr-2026

Contents

App Developer Framework

The Oracle Eloqua AppCloud developer framework is a complete development platform where you can easily build, register, and deploy apps for Eloqua. With new and improved service types which take advantage of Eloqua's bulk API, support for OAuth, and the ability to test your applications with Eloqua prior to launch, the Oracle Eloqua app developer framework provides the environment needed to put apps first.

The framework allows third parties to register and manage the integrations that they build for Eloqua right from their own Eloqua development instance. It also offers common, repeatable patterns for building apps that extend the functionality of Eloqua, providing a consistent, seamless experience for the marketing user.

Features

- App keys and OAuth2
- Test apps inside your development instance before going live
- Selective whitelististing so you can run betas on your apps before release
- Improved processing speeds with the bulk API
- A seamless flow for app installation and configuration
- 6 Oracle Eloqua app service types to support your different goals. For more details, see service types.

Flow

To get started with the Oracle Eloqua app developer framework:

1. Register as a provider in your Eloqua development instance. This gives you provider credentials that you can use to develop apps that interact with Eloqua.

2. Develop an app.
3. Register your app with Eloqua by providing a series of *templated URLs* that Eloqua uses to call out to your app. See the register services guide for details about what URL parameters you should include in each templated URL so that you will receive the data you need to do whatever your app does.
4. Publish the app to specific Eloqua instances or as a general release.
5. Whitelist a site and share your app's install URL so that the site's administrator can install the app and let marketers start using it.

Get started with the App Developer Framework

The Oracle Eloqua app developer framework is designed to make it easy to start working with the framework with minimal setup effort required. Once you have registered as a provider and ensured that your Eloqua user has sufficient permissions, you are ready to start developing an app!

Requirements

- An Eloqua instance in which you have registered as a provider.

> **Note**: If you are an external partner and want to publish your app on the Oracle CX Marketplace, see Becoming a Partner and Planning App Development to learn how to submit your app proposal before requesting an Eloqua Sandbox.

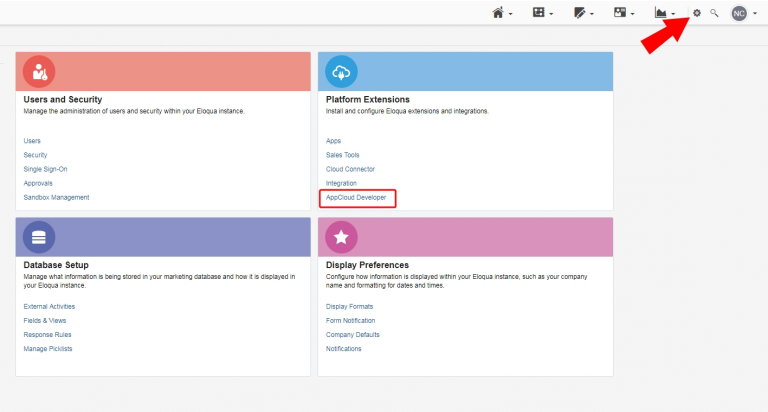
- Sufficient user permissions: the user making calls against Eloqua's APIs must have **Advanced Users - Marketing** permissions granted in the security groups section of the user management area of Eloqua.

Creating a provider

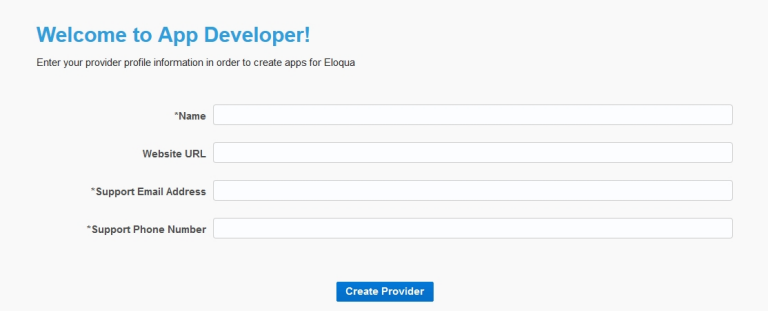
Learn more by watching the video

Registering as a provider

To register as a provider, log in to your Eloqua instance, and click **Settings** > **AppCloud Developer**.



Enter your company's information in the welcome page:

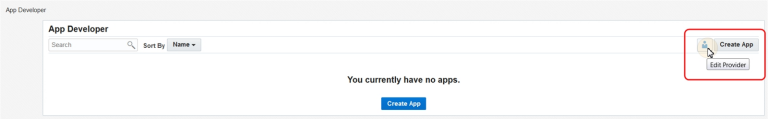


- **Name**: Your provider name. You might want to use your company name.
- **Website URL**: Your company website. This is optional.
- **Support email address**: The email address users should use to contact you if there are issues with your app.
- **Support phone number**: A phone number where you can be contacted if there are problems with your app.

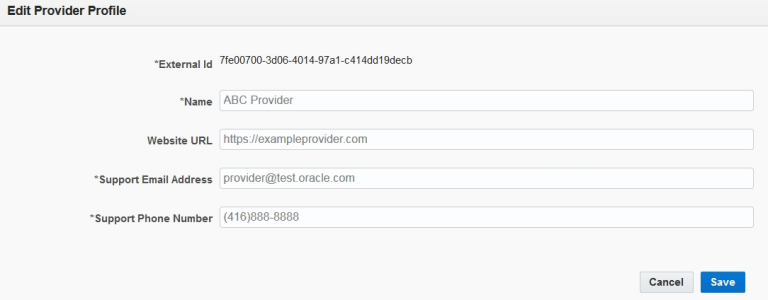
Click **Create Provider** > **Save**, and you're all set up as a provider. Your unique GUID-based provider **External ID** is displayed.

Edit provider information

You can edit this information by clicking **Edit Provider**.



The **Edit Provider Profile** page lists your provider information.



Next steps

Once you register as a provider, read the service descriptions to determine which ones you want to implement in your app, read about URL templating, and then consider the develop an app.

Becoming a Partner and Planning App Development

This topic explains how to become an Oracle Partner Network (OPN) member to start publishing apps on the Oracle Cloud Marketplace. On this page, *External partners* are defined as entities outside of Oracle who want to develop apps for the Eloqua App Framework.

The steps to plan an app are:

1. Evaluation & Onboarding
2. Access Developer Resources

3. Build and Test Your App
4. Review Your App
5. Release Your App

Step 1 - Evaluation & Onboarding

a. Submit the details of your proposed app using the evaluation form. If approved, the Eloqua Apps team will advise you to continue with the process.

b. Join the Oracle Partner Network. Applicable only to external partners. Oracle entities interested in developing generic apps for Eloqua on the App Framework can ignore this step.

> **Note**: Make sure to accept the Oracle CIA (Oracle Cloud Interoperability Appendum). This is a prerequsite for getting sandbox account access.

Step 2 - Access Developer Resources

a. Request access to an Eloqua Sandbox account (applicable only if you do not have your own Eloqua instance and you are an external partner who has already joined OPN and signed the CIA)

b. Join the Eloqua AppCloud Insiders Developer Community.

c. Access other useful developer resources:

- App Developer Framework Documentation
- Getting Started Videos
- Which Eloqua API Should I Use?
- Authentication
- Determining Base URLs

- Eloqua API Tutorials
- App Developer Framework Overview
- Changelog

Step 3 - Build and Test Your App

a. You are now ready to build your app. Make use of all the developer resources provided in the Eloqua Developer Help Center.

b. For any questions during app development, post them on the Eloqua AppCloud Insiders to get help from our tech experts and community members.

c. After you have developed your app and are ready to test, you must first Register your App with Eloqua.

Notes:

- The infrastructure for hosting the app is the responsibility of the App Provider.
- The implementation of the Framework services are optional.
- The use of OAuth Authentication to Eloqua APIs is required.

Step 4 - Review Your App

Before your submit your app for review, make sure you have thoroughly completed functional testing of your app.

a. Next, complete and send the Compatibility Review Document to the Eloqua Apps team. Please make sure you have taken care of all aspects covered in the document. As part of the review process, the Eloqua Apps team may ask for additional information and/or a demonstration of the app.

Step 5 - Release Your App

Step 5.1 Prepare App Resources

Prepare the resources for your app's release such as:

- Detailed user guide
- Pitch deck, demo videos etc.
- Customer on-boarding process
- Support model
- Enabling Oracle teams to showcase the app

Step 5.2 Publish Your App on the Marketplace

The Oracle Cloud Marketplace offers a platform for software vendors to show case their software that inter-operates with and extends Oracle CX Cloud (SaaS) Applications.

To publish your App on the Marketplace:

a. Become a Cloud Marketplace Publisher.
   - You must be an OPN member with an Oracle.com login. If you are an OPN member, but you do not have an Oracle.com login, follow the New User registration to associate your Oracle.com login with your OPN Membership Company ID.
   - Ensure to select only the Oracle products you are providing an integrated application.

b. Create a listing on the Cloud Marketplace following the Oracle Cloud Marketplace Listing Guidelines.
   i. Sign in to the Oracle Cloud Marketplace Partner Portal
   ii. Click **Home > Create Listing > Application Listing**.
   iii. In the **Demo URL** field, enter a video demonstration of your app hosted on YouTube or Vimeo.

iv. In the **Additional Information** tab from your app listing, complete and submit the following documents from the **Get Templates** section:

- Security Questionnaire, Test Plan
- Integrates-with Supplement - <Oracle CX Cloud Product> (required if your app is hosted on the Oracle Cloud)
- Partner Supplied Template: Deployment Guide (describes how to install, setup, and configure your application with the Oracle CX Cloud product)

Your listing may be returned for further editing to improve overall content or add/correct content. Reference the Marketplace Guidelines document for requirements and tips.

c. Publish your Marketplace listing

After the listing, video demonstration and technical documentation are submitted, your listing will be reviewed and approved by Oracle. You will receive a notice that your listing has been approved to publish. Now you can publish your listing within the Partner Portal!

i. Select your listing under **Submitted** and click **Publish**.

Service descriptions

Action services

Actions allow a marketer to add an action step to a campaign or program that is performed by an external system. For example, an action service could send contacts SMS messages, trigger a direct mailing, or register a contacts for a webinar. When contacts enter an action step, Eloqua calls out to the app, and the app responds by performing an action in an external system. The app then informs Eloqua that the process is complete and the contact moves on in the workflow. Read more about actions in the development guide and service registration instructions.

Learn more by watching the video

Decision services

Decisions allow a marketer to delegate a decision-making step in a campaign to an external system without having to bring the system's data into Eloqua. Decision services evaluate contacts using externally-held data, and determine whether the contacts meet the criteria. The service responds with yes or no to determine which path the contact should take through the campaign or program workflow. For example, a decision service could check if a contact exists in a CRM system, and sort the contacts accordingly. With Eloqua decision services, the data used in the decision never needs to be brought into Eloqua. Read more about decisions in the development guide and service registration instructions.

Learn more by watching the video

Feeder services

With feeders, developers can build apps that use data in third-party services to drive campaign and program membership. Feeders allow marketers to populate a campaign or program with contacts or custom objects from an external system, automatically importing the contacts or custom objects into Eloqua and adding them to a campaign or program. Read more about feeders in the development guide and service registration instructions.

> **Note**: For marketers using Eloqua's campaign canvas, feeder services are referred to as AppCloud Audiences.

Learn more by watching the video

Content services

Content allows marketers to source pieces of content in Eloqua emails and landing pages from an external source. This is the new and improved version of Eloqua's cloud components framework, and it includes many improvements such as asynchronous bulk processing (for emails), the ability to fully test the content service within Eloqua, and design-time interaction with the email and landing page editors. Read more about content in the development guide and service registration instructions.

Menu services

The AppCloud menus service is all about *context*. AppCloud menus enable a marketer to launch an externally-hosted application from within Eloqua via a menu dock. This menu dock floats on the right side of the screen in the campaign canvas or asset editor screens. Read more about menus in the development guide and service registration instructions.

Firehose services

The firehose service acts as a web hook and notifies an external system when a marketer performs an action in Eloqua. For example, when a marketer creates a campaign or edits an email, Eloqua can call out to the third-party with the update. Read more about the firehose service in the development guide and service registration instructions.

Introduction to URL templating

The Oracle Eloqua app developer framework supports *URL Templating* to enable you to configure the service URIs that Eloqua calls out to. Any acceptable template parameter is substituted with the appropriate value before the call is made.

Common URL parameters include `{UserId}`, the unique identifier for the user whose actions triggered the call, `{InstanceId}`, the GUID-based ID that identifies an instance of a service, and `{EventType}`, which provides campaign status information (created, activated, draft, and so on). These URL parameters enable you to configure the service URLs that Eloqua calls out to, specifying the information that you need for your application.

Example

Consider this templated URL:

https://awesomeapp.example.com/create/instance={InstanceId}&asset={AssetId}&site={SiteName}

When Eloqua calls the URL, the templated parameters are replaced with the appropriate values:

https://awesomeapp.example.com/create/instance=b5fc25ce-9709-42c4-a57d-caa00e23f658&asset=45&site=DocsExampleSite

Recommendations

Choose the URL parameters to include in your templated URLs based on your use case. These parameters provide contextual information: your application design will determine which parameters you need. The full list of parameters is available. The URL parameters reference provides a description for each parameter. You should familiarize yourself with the supported URL parameters when designing your templated URLs.

The service registration guides also include suggestions for parameters to include for specific URLs. These recommendations are the minimum you will likely need. Include additional parameters if you require them.

Develop an app

The Oracle Eloqua app developer framework provides 6 service types that you can develop to extend Eloqua's functionality. Some services create steps that a marketer can add to the campaign canvas, or to an email or landing page, another service type enables marketers to launch third-party applications directly from within Eloqua's UI, while another yet creates a web hook that calls out to a third-party service when a marketer performs an action within Eloqua.

These services make it easy for developers to extend Eloqua's capabilities, providing new opportunities for marketers to harness both Eloqua data and external services to better serve their goals.

Before you start developing Oracle Eloqua app services, familiarize yourself with the instantiation-execution model, which actions, decisions, feeders, and content follow. Then, you can read the development guides:

- Develop an action service
- Develop a decision service
- Develop a feeder service
- Develop a content service
- Develop a firehose service
- Develop a menu service

Instantiation-execution model

With the exception of menus and the firehose, Oracle Eloqua app services follow an instantiation-execution model. When you add an decision or action step to a campaign in the campaign canvas, a new instance of that decision or Action service is created. When that campaign or program reaches the decision or action stage, the service is executed. Similarly, when content is added to an email or landing page, a new instance of that content service is created.

Service instantiation

1. A Marketer drags the service onto a campaign or program canvas or onto an email or landing page asset. This triggers a call from the Eloqua UI to an internal Eloqua API that will interact with your servers.
2. The internal Eloqua API calls out to your service using the templated create URL you configured when registering your app. This is a POST request, authenticated with OAuth, which contains an empty JSON object. The call provides you with the new instance's GUID.

   For example, AwesomeApp has an AppCloud Decision service whose Create URL is:

   ```
   https://example.com/awesomeapp/decide/create?instance={instanceId}
   ```

   Eloqua calls out to this URL with an empty JSON object, replacing the templated parameters with the service instance ID and app installation ID.

   Your application should respond with default details in a Record Definition Data Transfer Object (DTO). For example, for a Decision service, the response might resemble:

   > **Note**: The "Content-Type" header must be set to "application/json" for the response to the Create URL call.

   ```
   HTTP/1.1 200 application/json; charset=utf-8


   {
    "recordDefinition": {
     "ContactID": "{{Contact.Id}}",
   ```

```
  "EmailAddress": "{{Contact.Field(C_EmailAddress)}}"
 }
}
```

This tells Eloqua to send the contacts' Contact ID and email address when the service is executed. For detailed instructions, see the documentation specific to each service type.

3. The marketer can then configure the instance using your service's UI by clicking **Edit** on the service instance. Eloqua opens a browser window to the configure URL for the service. This call to the configure URL comes directly from the marketing user's browser rather than from the Eloqua API. If you need to restrict access, you can do so here by requiring the user be on a specific VPN, for example.

4. If the Marketer has made changes, your API should call back to Eloqua using a `PUT` request with an updated DTO specifying the new record definition. For example, if the Marketer created an AppCloud Decision step that required `field1`, `field2`, and `field3`, the `PUT` request would resemble:

```
PUT https://secure.eloqua.com/api/cloud/1.0/decisions/instance/fddc932a-f27d-40c3-a126-82299b9512c5

{
  "recordDefinition":
  {
    "ContactID" : "{{Contact.Id}}",
    "EmailAddress" : "{{Contact.Field(C_EmailAddress)}}",
    "field1" : "{{Contact.Field(C_field1)}}",
    "field2" : "{{Contact.Field(C_field2)}}",
    "field3" : "{{Contact.Field(C_field3)}}"
  }
}
```

Execution

1. When the service is activated, either by contacts flowing through a campaign or a visit to a landing page, this batch of contacts is tagged with an execution ID that tracks its progress.
2. This step varies depending on the service type, but in general, the Eloqua internal API calls out to the service's notification URL, and transmits a the information you specified during the configuration phase. In the above example, then, the `recordDefinition` includes the contact's ID, email address, field1, field2 and field3. The call to the notification URL would then resemble:

```
POST https://example.com/awesomeapp/decide/notify?instance=fddc932a-f27d-40c3-a126-82299b9512c5&asset=456

{
 "offset" : 0,
 "limit" : 1000,
 "totalResults" : 2,
 "count" : 2,
 "hasMore" : false,
 "items" :
 [
   {
     "ContactID" : "1",
     "EmailAddress" : "fred@example.com",
     "field1" : "stuff",
     "field2" : "things",
     "field3" : "et cetera"
   },
   {
     "ContactID" : "2",
```

```
      "EmailAddress" : "sylvie@example.com",
      "field1" : "more stuff",
      "field2" : "other things",
      "field3" : "and so on"
    }
  ]
}
```

This follows the format defined during the create URL and configure URL calls during the instantiation phase.

3. If Eloqua calls out to your app and doesn't receive a response within 100 seconds, the contacts in a step will be marked as "Errored". If Eloqua receives a response status code that is between 300 and 599, Eloqua will retry the notify call over approximately an eight-hour period of time with a backoff strategy of the time between calls doubling after each call. After this eight-hour period, the contacts in a step will be marked as "Errored". If the marketer has configured a default path for the contacts to flow through, then the contacts will flow into the next step.
4. If you specified maximum record size of 0 when you configured your app, or the record definition sent during the instantiation phase contains no fields, the DTO contains an empty JSON object. Otherwise, the call sends a DTO containing contact data, with a maximum limit to the number of records as specified in the service configuration. If there are more records than the maximum limit, Eloqua will keep sending requests until all the data is transmitted.
5. You can then choose to return the same response for all the contacts in the batch, or tailor your responses to each contact.

   To individualize your responses for each contact, the desired HTTP response status code is a 204, followed up with an import call using the bulk API. You must specify sync actions during the import. The appropriate sync actions vary according to the service type. Refer to the bulk API documentation and the service-specific documentation for more information

6. Eloqua receives the import. For service instances associated with a campaign, those contacts flow through to the next stage.

Notification URL setup

Many AppCloud services require that you configure a Notification URL when registering the service. The Notification URL is essentially a webhook: it is the URL that Eloqua should call out to when something happens.

For example, when an Eloqua contact visits a landing page containing an AppCloud content service instance, Eloqua calls out to the service provider's Notification URL to request the HTML content needed to populate the landing page. Similarly, when contacts flow into an AppCloud decision or AppCloud action step during a campaign, Eloqua calls out to the notification URL to notify the app that contacts have reached that step.

The notification URL is a templated URL which supports the common parameters, including `{InstanceId}`, `{InstallId}`, `{AssetId}`, `{AssetName}`, `{UserName}`, `{UserId}`, `{EventType}`, and `{UserCulture}`. The AppCloud content service notification URL also supports `{VisitorId}`, which maps to the ID of a visitor when called from a landing page, to enable delivery of visitor-specific content.

When the notification URL is called for AppCloud actions, decisions, feeders, or content, Eloqua also sends, as parameters, the fields specified by your app during the instantiation phase. For example, AwesomeApp has an AppCloud decision service. Its notification URL is:

```
https://example.com/awesomeapp/decide/notify?instance=
{InstanceId}&asset={AssetId}
```

During instantiation, ExampleApp specified that it requires the contact ID, email address, field1, field2, and field3 in the *recordDefinition* field of the service instance data transfer object. The *recordDefinition* field was:

```
"recordDefinition": {
  "contactID": "{{Contact.Id}}",
  "email": "{{Contact.Field(C_EmailAddress)}}",
```

```
  "field1": "{{Contact.Field(C_Field1)}}",
  "field2": "{{Contact.Field(C_Field2)}}",
  "field3": "{{Contact.Field(C_Field3)}}"
}
```

Suppose that two contacts, fred@example.com and sylvie@example.com, flowed into an AppCloud decision step. Eloqua's call to the notification URL would then resemble:

```
POST https://example.com/awesomeapp/decide/notify?instance=123&asset=456
{
  "offset" : 0,
  "limit" : 1000,
  "totalResults" : 2,
  "count" : 2,
  "hasMore" : false,
  "items" :
  [
    {
     "contactID" : "1",
     "email" : "fred@example.com",
     "field1" : "stuff",
     "field2" : "things",
     "field3" : "et cetera"
    },
    {
     "contactID" : "2",
     "email" : "sylvie@example.com",
     "field1" : "more stuff",
     "field2" : "other things",
     "field3" : "and so on"
    }
   ]
}
```

The appropriate response to the notification URL call varies slightly depending on which service you're developing. You can see the appropriate responses for each service type in the following documents:

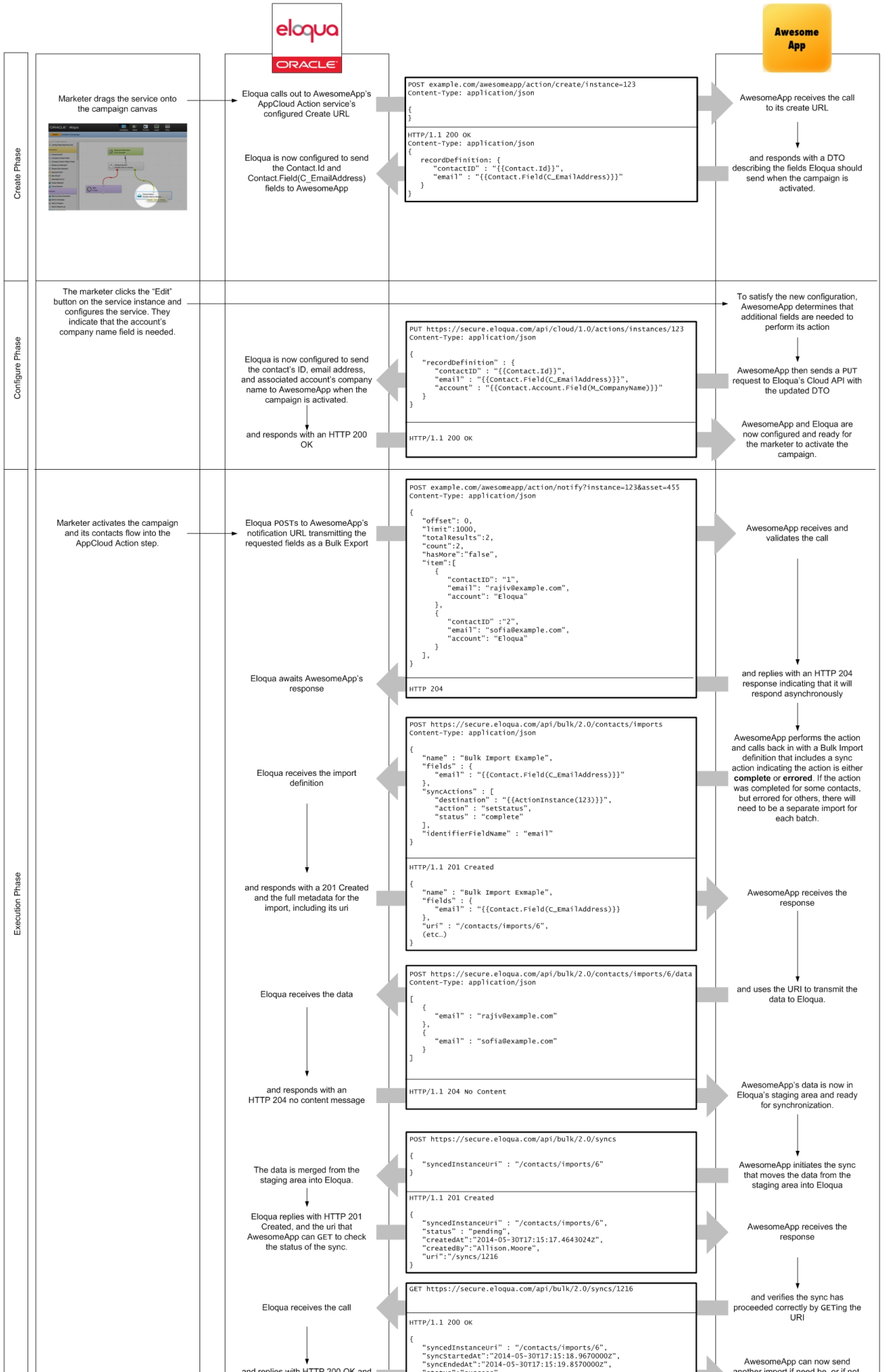
- Develop an AppCloud action service
- Develop an AppCloud decision service
- Develop an AppCloud content service
- Develop an AppCloud feeder service
- Develop an AppCloud firehose service

Develop an Oracle Eloqua app action service

Actions are steps on a campaign or program canvas that are delegated to an external system. This allows you to perform actions that you cannot do with Eloqua. Actions are analogous to the existing cloud connectors.

Actions follow the instantiation-execution model. The following sections describe action-specific details and provide the endpoints needed to develop an Action service. If you are not familiar with the general flow for the instantiation-execution model, you should read that first.

Eloqua signs all outgoing calls with OAuth 1.0a so the receiving system can validate that the call was sent by Eloqua. Refer to the OAuth 1.0a spec or OAuth 1.0 RFC for more information.


eloqua
ORACLE
Awesome App
Create Phase
Marketer drags the service onto the campaign canvas
Eloqua calls out to AwesomeApp's AppCloud Action service's configured Create URL
POST example.com/awesomeapp/action/create/instance=123
Content-Type: application/json
{
}
HTTP/1.1 200 OK
Content-Type: application/json
{
recordDefinition: {
"contactID" : "{{Contact.Id}}",
"email" : "{{Contact.Field(C_EmailAddress)}}"
}
}
AwesomeApp receives the call to its create URL
and responds with a DTO describing the fields Eloqua should send when the campaign is activated.
Eloqua is now configured to send the Contact.Id and Contact.Field(C_EmailAddress) fields to AwesomeApp
Configure Phase
The marketer clicks the "Edit" button on the service instance and configures the service. They indicate that the account's company name field is needed.
To satisfy the new configuration, AwesomeApp determines that additional fields are needed to perform its action
PUT https://secure.eloqua.com/api/cloud/1.0/actions/instances/123
Content-Type: application/json
{
"recordDefinition" : {
"contactID" : "{{Contact.Id}}",
"email" : "{{Contact.Field(C_EmailAddress)}}",
"account" : "{{Contact.Account.Field(M_CompanyName)}}"
}
}
HTTP/1.1 200 OK
AwesomeApp then sends a PUT request to Eloqua's Cloud API with the updated DTO
Eloqua is now configured to send the contact's ID, email address, and associated account's company name to AwesomeApp when the campaign is activated.
and responds with an HTTP 200 OK
AwesomeApp and Eloqua are now configured and ready for the marketer to activate the campaign.
Execution Phase
Marketer activates the campaign and its contacts flow into the AppCloud Action step.
Eloqua POSTs to AwesomeApp's notification URL transmitting the requested fields as a Bulk Export
POST example.com/awesomeapp/action/notify?instance=123&asset=455
Content-Type: application/json
{
"offset": 0,
"limit":1000,
"totalResults":2,
"count":2,
"hasMore":"false",
"item":[
{
"contactID": "1",
"email": "rajiv@example.com",
"account": "Eloqua"
},
{
"contactID" :"2",
"email": "sofia@example.com",
"account": "Eloqua"
}
],
}
HTTP 204
AwesomeApp receives and validates the call
and replies with an HTTP 204 response indicating that it will respond asynchronously
Eloqua awaits AwesomeApp's response
POST https://secure.eloqua.com/api/bulk/2.0/contacts/imports
Content-Type: application/json
{
"name" : "Bulk Import Example",
"fields" : {
"email" : "{{Contact.Field(C_EmailAddress)}}"
},
"syncActions" : [
"destination" : "{{ActionInstance(123)}}",
"action" : "setStatus",
"status" : "complete"
],
"identifierFieldName" : "email"
}
HTTP/1.1 201 Created
{
"name" : "Bulk Import Exmaple",
"fields" : {
"email" : "{{Contact.Field(C_EmailAddress)}}
},
"uri" : "/contacts/imports/6",
(etc…)
}
AwesomeApp performs the action and calls back in with a Bulk Import definition that includes a sync action indicating the action is either complete or errored. If the action was completed for some contacts, but errored for others, there will need to be a separate import for each batch.
Eloqua receives the import definition
and responds with a 201 Created and the full metadata for the import, including its uri
AwesomeApp receives the response
POST https://secure.eloqua.com/api/bulk/2.0/contacts/imports/6/data
Content-Type: application/json
[
{
"email" : "rajiv@example.com"
},
{
"email" : "sofia@example.com"
}
]
HTTP/1.1 204 No Content
Eloqua receives the data
and responds with an HTTP 204 no content message
and uses the URI to transmit the data to Eloqua.
AwesomeApp's data is now in Eloqua's staging area and ready for synchronization.
POST https://secure.eloqua.com/api/bulk/2.0/syncs
{
"syncedInstanceUri" : "/contacts/imports/6"
}
HTTP/1.1 201 Created
{
"syncedInstanceUri" : "/contacts/imports/6",
"status" : "pending",
"createdAt":"2014-05-30T17:15:17.4643024Z",
"createdBy":"Allison.Moore",
"uri":"/syncs/1216
}
The data is merged from the staging area into Eloqua.
Eloqua replies with HTTP 201 Created, and the uri that AwesomeApp can GET to check the status of the sync.
AwesomeApp initiates the sync that moves the data from the staging area into Eloqua
AwesomeApp receives the response
GET https://secure.eloqua.com/api/bulk/2.0/syncs/1216
HTTP/1.1 200 OK
{
"syncedInstanceUri" : "/contacts/imports/6",
"syncStartedAt":"2014-05-30T17:15:18.9670000Z",
"syncEndedAt":"2014-05-30T17:15:19.8570000Z",
Eloqua receives the call
and replies with HTTP 200 OK and
and verifies the sync has proceeded correctly by GETing the URI
AwesomeApp can now send

Create URL

Eloqua's call to the **CreateURL**:

```
POST https://example.com/awesomeapp/act/create?instance=f82d50cd-86a9-4fca-b37e-4ec9a98b0339

{}
```

AwesomeApp's response is a DTO describing the fields Eloqua should transmit when the service in executed. You can include a maximum of 249 fields in your record definition.

> **Note**: The "Content-Type" header must be set to "application/json" for the response to the Create URL call.

Example response for contacts:

```
HTTP/1.1 200 application/json; charset=utf-8

{
 "recordDefinition": {
  "ContactID": "{{Contact.Id}}",
  "EmailAddress": "{{Contact.Field(C_EmailAddress)}}"
 },
 "requiresConfiguration": true
}
```

Example response for custom objects:

```
HTTP/1.1 200 application/json; charset=utf-8

{
 "recordDefinition": {
  "Email": "{{CustomObject[1].Field[4]}}"
 },
 "requiresConfiguration": true
}
```

`requiresConfiguration` is an optional Boolean parameter which tells Eloqua whether user configuration is required before the app can be used. If set to `true`, users will be unable to activate a campaign or program containing the unconfigured app service instance. Eloqua will display an error message.

❏**Warning**: All field names and values in Eloqua data transfer objects are case sensitive. Be sure to reproduce them exactly. For example: `{{Contact.id}}` would fail if the correct form is `{{Contact.Id}}`.

Configure URL

When a user clicks the edit button on the service instance and makes changes through the **Configure URL** that require an updated recordDefinition data transfer object (DTO), AwesomeApp must call out to Eloqua's cloud API `PUT /actions/instances/{id}` endpoint with that updated DTO:

Example request for contacts:

```
PUT https://secure.p03.eloqua.com/api/cloud/1.0/actions/instances/f82d50cd-86a9-4fca-b37e-4ec9a98b0339

{
   "recordDefinition":
     {
       "ContactID" : "{{Contact.Id}}",
       "EmailAddress" : "{{Contact.Field(C_EmailAddress)}}",
       "field1" : "{{Contact.Field(C_field1)}}",
       "field2" : "{{Contact.Field(C_field2)}}",
       "field3" : "{{Contact.Field(C_field3)}}"
     },
   "requiresConfiguration": false
}
```

Example request for custom objects:

```
PUT https://secure.p03.eloqua.com/api/cloud/1.0/actions/instances/f82d50cd-86a9-4fca-b37e-4ec9a98b0339

{
   "recordDefinition": {
        "Email": "{{CustomObject[1].Field[4]}}"
   },
  "requiresConfiguration": false
}
```

`requiresConfiguration` is an optional Boolean parameter which tells Eloqua whether user configuration is required before the app can be used. If set to `true`, users will be unable to activate a campaign or program containing the unconfigured app service instance. Eloqua will display an error message.

Use the Configure URL response to set `requiresConfiguration` to `false` when your app's configuration acceptance criteria have been met.

Note: `X-Frame-Options` must not be set to `DENY` for any configure page.

Warning: If the campaign or program is not in draft mode, attempting to set `requiresConfiguration` to `true` will result in an error.

Notification URL

Eloqua calls the **Notification URL** when contacts or custom objects arrive in the action step. This call transmits the requested fields in the `items` parameter:

```
POST https://example.com/awesomeapp/act/notify?instance=f82d50cd-86a9-4fca-b37e-4ec9a98b0339&asset=456

{
   "offset" : 0,
```

```
  "limit" : 1000,
  "totalResults" : 2,
  "count" : 2,
  "hasMore" : false,
  "items" :
  [
    {
      "ContactID" : "1",
      "EmailAddress" : "fred@example.com",
      "field1" : "stuff",
      "field2" : "things",
      "field3" : "et cetera"
    },
    {
      "ContactID" : "2",
      "EmailAddress" : "john@example.com",
      "field1" : "more stuff",
      "field2" : "other things",
      "field3" : "and so on"
    }
  ]
}
```

Your app must respond to this call, otherwise Eloqua will think the call has timed out.

Note: The amount of records in the `items` parameter is dependent on the **Records per Notification** option during service registration. If you prefer to retrieve records using a separate export, see Retrieving app records using the bulk API for instructions.

Important: If AwesomeApp's Action service is configured to automatically set a contact or custom object record's status to complete, contacts and custom object records will move to the next step in the canvas after a successful response from the app to the notification URL call. Each batch, delivered through a call to the notification URL, of contacts or custom object records in a step will only remain in the workflow for a maximum of 90 days. Attempts to operate on them (set the status

to complete or errored) after 90 days will fail, resulting in the records being stuck in the step, and requiring manual intervention to move them along in the flow. Each batch of records sent through a call to the notification URL has an ExecutionId, available as a template parameter, that identifies a unique batch.

If the action service is not configured to automatically set the status to complete, AwesomeApp should return a 204 response. This indicates that the response will be asynchronous. AwesomeApp should then create an import to Eloqua using the Bulk API where AwesomeApp specifies a sync action: either complete or errored.

Bulk API contact import

1. Create the bulk import definition, setting the status to complete to import data.

   If there is no data to import, and you only need to update a record's status, you can update a record's status without performing an import by creating a contact sync action definition.

   When importing, the destination field refers to the action service's instance. In this example, the instance GUID is `f82d50cd-86a9-4fca-b37e-4ec9a98b0339`:

   > **Warning**: When referencing service instances, you must transmit the GUID without dashes. The bulk API will error if you transmit the GUID with the dashes.

   Create a contact import definition where the sync action sets the record's status, where the instance ID is `f82d50cd-86a9-4fca-b37e4ec9a98b0339` and execution ID is `12345`:

   ```
   POST https://secure.p03.eloqua.com/api/bulk/2.0/contacts/imports
   ```

```
{
  "name" : "AwesomeApp Action Response Bulk Import",
  "updateRule" : "always",
  "fields" : {
    "emailAddress" : "{{Contact.Field(C_EmailAddress)}}"
  },
  "syncActions" : [
    {
      "destination" : "{{ActionInstance
(f82d50cd86a94fcab37e4ec9a98b0339).Execution[12345]}}",
      "action" : "setStatus",
      "status" : "complete"
    }
  ],
  "identifierFieldName" : "emailAddress"
}
```

Eloqua's response will be a 201 created response that includes a `uri` parameter, which you can use to identify the import:

```
  {
      "name" : "AwesomeApp Action Response Bulk Import",
      "updateRule" : "always",
      "fields" : {
        "emailAddress" : "{{Contact.Field(C_EmailAddress)}}"
      },
      "identifierFieldName" : "emailAddress",
      "syncActions" : [
        {
```

```
        "destination" : "{{ActionInstance
  (f82d50cd86a94fcab37e4ec9a98b0339).Execution[12345]}}",
        "action" : "setStatus",
        "status" : "complete"
      }
    ],
    "isSyncTriggeredOnImport" : false,
    "isUpdatingMultipleMatchedRecords" : false,
    "uri" : "/contacts/imports/6",
    "createdBy" : "DocsExample",
    "createdAt" : "2014-03-06T13:59:00.6600046Z",
    "updatedBy" : "DocsExample",
    "updatedAt" : "2014-03-06T13:59:00.6600046Z"
  }
```

2. Send Eloqua the import data using the `uri`:

```
POST https://secure.p03.eloqua.com/api/bulk/2.0/contacts/imports/6/data
```

```
[
   {
     "emailAddress" : "fred@example.com"
   },
   {
     "emailAddress" : "sylvie@example.com"
   }
]
```

3. Synchronize the data for import:

   AwesomeApp's request:

```
POST https://secure.p03.eloqua.com/api/bulk/2.0/syncs

{
   "syncedInstanceURI":"/contacts/imports/6"
}
```

Eloqua's response:

```
201 Created


{
    "syncedInstanceURI" : "/contacts/imports/6",
    "status" : "pending",
    "createdAt" : "2014-01-01T13:59:07.1375620Z",
    "createdBy" : "DocsExample",
    "uri" : "/syncs/6"
}
```

4. You can then use the sync's URI (`/syncs/6`) to check the status of the sync:

```
GET https://secure.p03.eloqua.com/api/bulk/2.0/sync/6
```

When the sync is complete, Eloqua's response will resemble:

```
200 OK


{
     "syncedInstanceURI" : "/contacts/imports/6",
     "syncStartedAt" : "2014-01-01T13:59:07.1375620Z",
     "syncEndedAt" : "2014-01-01T13:59:14.1375620Z",
     "status" : "success",
```

```
    "createdAt" : "2014-01-01T13:59:07.1375620Z",
    "createdBy" : "DocsExample",
    "uri" : "/syncs/6"
}
```

> **Learn more**: Bulk API imports.

Bulk API custom object import

1. Create the bulk import definition, setting the status to complete to import data.

   If there is no data to import, and you only need to update a record's status, you can update a record's status without performing an import by creating a custom object sync action definition.

   When importing, the destination field refers to the action service's instance - in this example, the instance GUID is `f82d50cd-86a9-4fca-b37e-4ec9a98b0339`:

   > **Warning**: When referencing service instances, you must transmit the GUID without dashes. The bulk API will error if you transmit the GUID with the dashes.

   Create a custom object import definition where the sync action sets the record's status, where the instance ID is `f82d50cd-86a9-4fca-b37e4ec9a98b0339` and execution ID is `12345`:

```
POST https://secure.p03.eloqua.com/api/bulk/2.0/customObjects/9/imports

{
  "name" : "AwesomeApp Action Response Bulk Import",
```

```
  "updateRule" : "always",
  "fields" : {
    "email" : "{{CustomObject[9].Field[58]}}"
  },
  "syncActions" : [
    {
      "destination" : "{{ActionInstance
(f82d50cd86a94fcab37e4ec9a98b0339).Execution[12345]}}",
      "action" : "setStatus",
      "status" : "complete"
    }
  ],
  "identifierFieldName" : "email"
}
```

Eloqua's response will be a 201 created response that includes a `uri` parameter, which you can use to identify the import:

```
  HTTP/1.1 201 Created

  {
      "name" : "AwesomeApp Action Response Bulk Import",
      "updateRule" : "always",
      "fields" : {
        "email" : "{{CustomObject[9].Field[58]}}"
      },
      "identifierFieldName" : "email",
      "syncActions" : [
        {
          "destination" : "{{ActionInstance
```

```
  (f82d50cd86a94fcab37e4ec9a98b0339).Execution[12345]}}",
        "action" : "setStatus",
        "status" : "complete"
      }
    ],
    "isSyncTriggeredOnImport" : false,
    "isUpdatingMultipleMatchedRecords" : false,
    "uri" : "/customObjects/imports/9",
    "createdBy" : "DocsExample",
    "createdAt" : "2014-03-06T13:59:00.6600046Z",
    "updatedBy" : "DocsExample",
    "updatedAt" : "2014-03-06T13:59:00.6600046Z"
  }
```

2. Send Eloqua the data for import as a using the `uri`:

```
POST https://secure.p03.eloqua.com/api/bulk/2.0/customObjects/imports/9/data
```

```
[
   {
     "email" : "fred@example.com"
   },
   {
     "email" : "sylvie@example.com"
   }
]
```

3. Synchronize the data for import:

   AwesomeApp request

```
POST https://secure.p03.eloqua.com/api/bulk/2.0/syncs

{
   "syncedInstanceURI":"/customObjects/imports/9"
}
```

Eloqua's response:

```
201 Created


{
    "syncedInstanceURI" : "/customObjects/imports/9",
    "status" : "pending",
    "createdAt" : "2014-01-01T13:59:07.1375620Z",
    "createdBy" : "DocsExample",
    "uri" : "/syncs/9"
}
```

4. You can then use the sync's uri (`/syncs/9`) to check the status of the sync:

```
GET https://secure.p03.eloqua.com/api/bulk/2.0/sync/9
```

When the sync is complete, Eloqua's response will resemble:

```
200 OK


{
     "syncedInstanceURI" : "/customObjects/imports/9",
     "syncStartedAt" : "2014-01-01T13:59:07.1375620Z",
     "syncEndedAt" : "2014-01-01T13:59:14.1375620Z",
     "status" : "success",
```

```
    "createdAt" : "2014-01-01T13:59:07.1375620Z",
    "createdBy" : "DocsExample",
    "uri" : "/syncs/9"
}
```

Learn more: Bulk API imports.

Delete URL

The delete URL is a templated URL pointing to an endpoint for deleting an instance of your service.

The delete URL uses an HTTP DELETE request and there is no content sent in the request body. All common URL template parameters are available (the same as with a create URL). On success, this endpoint should return a 200-level response.

An example delete URL would look something like:

```
https://www.someurl.com/delete/{appId}/{installId}/{instanceId}/{userName}/{siteName}/{siteId}
```

Note: Delete calls are not sent in real time, but are done in a batch once daily.

Learn more

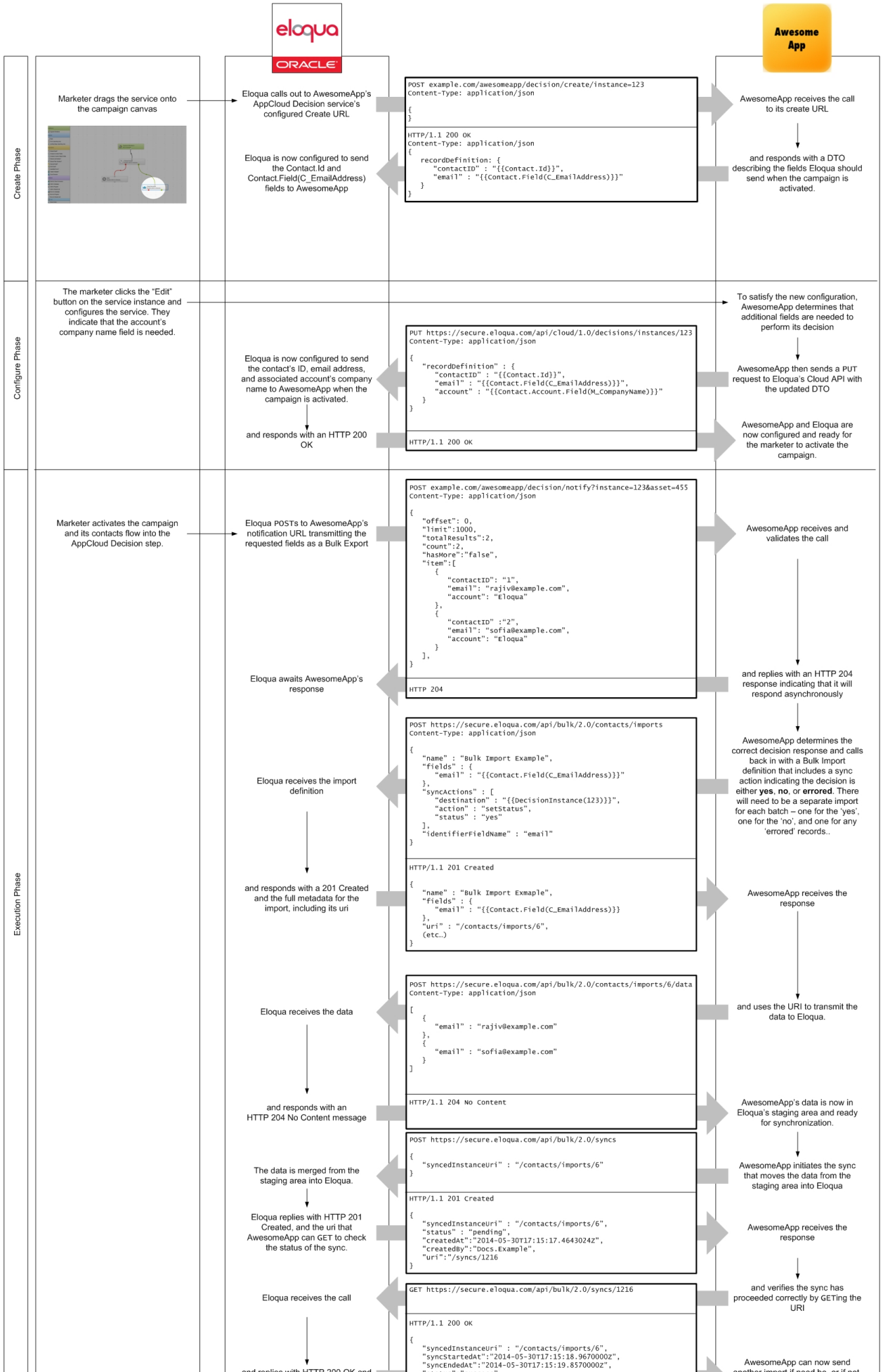
Develop an Oracle Eloqua app decision service

Decisions are steps on a campaign or program canvas that directly control the path a contact or custom object takes as they flow through a canvas. Decision steps delegate the decision making to an external system, which responds with a yes or no for each

object flowing through the canvas, determining which path the contact or custom object should take.

Decisions follow the instantiation-execution model. The following sections describe decision-specific details and provide the endpoints needed to develop a decision service. If you are not familiar with the general flow for the instantiation-execution model, you should read that first.

Eloqua signs all outgoing calls with OAuth 1.0a so the receiving system can validate that the call was sent by Eloqua. Refer to the OAuth 1.0a spec or OAuth 1.0 RFC for more information.


eloqua
ORACLE
Awesome App
Create Phase
Marketer drags the service onto the campaign canvas
Eloqua calls out to AwesomeApp's AppCloud Decision service's configured Create URL
POST example.com/awesomeapp/decision/create/instance=123
Content-Type: application/json
{
}
AwesomeApp receives the call to its create URL
HTTP/1.1 200 OK
Content-Type: application/json
{
recordDefinition: {
"contactID" : "{{Contact.Id}}",
"email" : "{{Contact.Field(C_EmailAddress)}}"
}
}
and responds with a DTO describing the fields Eloqua should send when the campaign is activated.
Eloqua is now configured to send the Contact.Id and Contact.Field(C_EmailAddress) fields to AwesomeApp
Configure Phase
The marketer clicks the "Edit" button on the service instance and configures the service. They indicate that the account's company name field is needed.
To satisfy the new configuration, AwesomeApp determines that additional fields are needed to perform its decision
PUT https://secure.eloqua.com/api/cloud/1.0/decisions/instances/123
Content-Type: application/json
{
"recordDefinition" : {
"contactID" : "{{Contact.Id}}",
"email" : "{{Contact.Field(C_EmailAddress)}}",
"account" : "{{Contact.Account.Field(M_CompanyName)}}"
}
}
Eloqua is now configured to send the contact's ID, email address, and associated account's company name to AwesomeApp when the campaign is activated.
AwesomeApp then sends a PUT request to Eloqua's Cloud API with the updated DTO
HTTP/1.1 200 OK
and responds with an HTTP 200 OK
AwesomeApp and Eloqua are now configured and ready for the marketer to activate the campaign.
Execution Phase
Marketer activates the campaign and its contacts flow into the AppCloud Decision step.
Eloqua POSTs to AwesomeApp's notification URL transmitting the requested fields as a Bulk Export
POST example.com/awesomeapp/decision/notify?instance=123&asset=455
Content-Type: application/json
{
"offset": 0,
"limit":1000,
"totalResults":2,
"count":2,
"hasMore":"false",
"item":[
{
"contactID": "1",
"email": "rajiv@example.com",
"account": "Eloqua"
},
{
"contactID" :"2",
"email": "sofia@example.com",
"account": "Eloqua"
}
],
}
AwesomeApp receives and validates the call
HTTP 204
Eloqua awaits AwesomeApp's response
and replies with an HTTP 204 response indicating that it will respond asynchronously
POST https://secure.eloqua.com/api/bulk/2.0/contacts/imports
Content-Type: application/json
{
"name" : "Bulk Import Example",
"fields" : {
"email" : "{{Contact.Field(C_EmailAddress)}}"
},
"syncActions" : [
"destination" : "{{DecisionInstance(123)}}",
"action" : "setStatus",
"status" : "yes"
],
"identifierFieldName" : "email"
}
Eloqua receives the import definition
AwesomeApp determines the correct decision response and calls back in with a Bulk Import definition that includes a sync action indicating the decision is either **yes**, **no**, or **errored**. There will need to be a separate import for each batch – one for the 'yes', one for the 'no', and one for any 'errored' records..
HTTP/1.1 201 Created
{
"name" : "Bulk Import Exmaple",
"fields" : {
"email" : "{{Contact.Field(C_EmailAddress)}}
},
"uri" : "/contacts/imports/6",
(etc…)
}
and responds with a 201 Created and the full metadata for the import, including its uri
AwesomeApp receives the response
POST https://secure.eloqua.com/api/bulk/2.0/contacts/imports/6/data
Content-Type: application/json
[
{
"email" : "rajiv@example.com"
},
{
"email" : "sofia@example.com"
}
]
Eloqua receives the data
and uses the URI to transmit the data to Eloqua.
HTTP/1.1 204 No Content
and responds with an HTTP 204 No Content message
AwesomeApp's data is now in Eloqua's staging area and ready for synchronization.
POST https://secure.eloqua.com/api/bulk/2.0/syncs
{
"syncedInstanceUri" : "/contacts/imports/6"
}
The data is merged from the staging area into Eloqua.
AwesomeApp initiates the sync that moves the data from the staging area into Eloqua
HTTP/1.1 201 Created
{
"syncedInstanceUri" : "/contacts/imports/6",
"status" : "pending",
"createdAt":"2014-05-30T17:15:17.4643024Z",
"createdBy":"Docs.Example",
"uri":"/syncs/1216
}
Eloqua replies with HTTP 201 Created, and the uri that AwesomeApp can GET to check the status of the sync.
AwesomeApp receives the response
GET https://secure.eloqua.com/api/bulk/2.0/syncs/1216
Eloqua receives the call
and verifies the sync has proceeded correctly by GETing the URI
HTTP/1.1 200 OK
{
"syncedInstanceUri" : "/contacts/imports/6",
"syncStartedAt":"2014-05-30T17:15:18.9670000Z",
"syncEndedAt":"2014-05-30T17:15:19.8570000Z",
AwesomeApp can now send

Create URL

Eloqua's call to the **CreateURL**:

```
POST https://example.com/awesomeapp/decide/create?instance=9347bfe1-9c72-409c-a5cd-402ff74f0caa

{}
```

AwesomeApp's response is a DTO describing the fields Eloqua should transmit when the service in executed. You can include a maximum of 249 fields in your record definition.

> **Note**: The "Content-Type" header must be set to "application/json" for the response to the Create URL call.

Example response for contacts:

```
HTTP/1.1 200 application/json; charset=utf-8

{
  "recordDefinition": {
    "ContactID": "{{Contact.Id}}",
    "EmailAddress": "{{Contact.Field(C_EmailAddress)}}"
  },
  "requiresConfiguration": true
}
```

Example response for custom objects:

```
HTTP/1.1 200 application/json; charset=utf-8

{
  "recordDefinition": {
    "Email": "{{CustomObject[1].Field[4]}}"
  },
  "requiresConfiguration": true
}
```

`requiresConfiguration` is an optional Boolean parameter which tells Eloqua whether user configuration is required before the app can be used. If set to `true`, users will be unable to activate a campaign or program containing the unconfigured app service instance. Eloqua will display an error message.

❏**Warning**: All field names and values in Eloqua data transfer objects are case sensitive. Be sure to reproduce them exactly. For example: `{{Contact.id}}` would fail if the correct form is `{{Contact.Id}}`.

Configure URL

When a user clicks the edit button on the service instance and makes changes through the **Configure URL** that require an updated recordDefinition DTO, AwesomeApp must call out to Eloqua's cloud API `PUT /decisions/instances/{id}` endpoint with that updated DTO:

Example request for contacts:

```
PUT https://secure.p03.eloqua.com/api/cloud/1.0/decisions/instances/9347bfe1-9c72-409c-a5cd-402ff74f0caa

{
   "recordDefinition":
     {
       "ContactID" : "{{Contact.Id}}",
       "EmailAddress" : "{{Contact.Field(C_EmailAddress)}}",
       "field1" : "{{Contact.Field(C_field1)}}",
       "field2" : "{{Contact.Field(C_field2)}}",
       "field3" : "{{Contact.Field(C_field3)}}"
     },
   "requiresConfiguration": false
}
```

Example request for custom objects:

```
PUT https://secure.p03.eloqua.com/api/cloud/1.0/decisions/instances/9347bfe1-9c72-409c-a5cd-402ff74f0caa

{
   "recordDefinition": {
        "Email": "{{CustomObject[1].Field[4]}}"
   },
  "requiresConfiguration": false
}
```

`requiresConfiguration` is an optional Boolean parameter which tells Eloqua whether user configuration is required before the app can be used. If set to `true`, users will be unable to activate a campaign or program containing the unconfigured app service instance. Eloqua will display an error message.

Use the Configure URL response to set `requiresConfiguration` to `false` when your app's configuration acceptance criteria have been met.

> **Note**: `X-Frame-Options` must not be set to `DENY` for any configure page.

> **Warning**: If the campaign or program is not in draft mode, attempting to set `requiresConfiguration` to `true` will result in an error.

Notification URL

Eloqua's call to the **Notification URL** transmits the requested fields in the items parameter:

```
POST https://example.com/awesomeapp/decide/notify?instance=9347bfe1-9c72-409c-a5cd-402ff74f0caa&asset=456

{
   "offset" : 0,
   "limit" : 1000,
```

```
  "totalResults" : 2,
  "count" : 2,
  "hasMore" : false,
  "items" :
  [
    {
      "ContactID" : "1",
      "EmailAddress" : "fred@example.com",
      "field1" : "stuff",
      "field2" : "things",
      "field3" : "et cetera"
    },
    {
      "ContactID" : "2",
      "EmailAddress" : "john@example.com",
      "field1" : "more stuff",
      "field2" : "other things",
      "field3" : "and so on"
    }
  ]
}
```

Note: The amount of records in the `items` parameter is dependent on the **Records per Notification** option during service registration. If you prefer to retrieve records using a separate export, see Retrieving app records using the bulk API for instructions.

Important: Each batch, delivered through a call to the notification URL, of contacts or custom object records in a step will only remain in the workflow for a maximum of 90 days. Attempts to operate on them (set the status to complete or errored) after 90 days will fail, resulting in the records being stuck in the step, and requiring manual intervention to move them along in the flow. Each batch of records sent through a call to the notification URL has an ExecutionId, available as a template parameter, that identifies a unique batch.

AwesomeApp responds with a 204 response, indicating that the response will be asynchronous, followed by an **import** to Eloqua's Bulk API, where AwesomeApp specifies a sync action. For Decisions, the sync action is either "**yes**", "**no**", or "**errored**". AwesomeApp will need to create multiple imports for the different statuses, with contacts or custom objects that should follow the "no" path in one import and then contacts and custom objects that should follow the "yes" path in another. Note that the maximum import size per batch is 5,000: if you have more than 5,000 contacts or custom objects, break your data up into multiple imports.

Bulk API contact import

1. Create the bulk import definition, setting the status to **yes** to import data.

   If there is no data to import, and you only need to update a record's status, you can update a record's status without performing an import by creating a contact sync action definition.

   When importing, the "destination" field refers to the AppCloud Decision service's instance - in this example, the instance ID is `9347bfe1-9c72-409c-a5cd-402ff74f0caa`:

   > **Warning**: When referencing service instances, you must transmit the GUID without dashes. The bulk API will error if you transmit the GUID with the dashes.

   Create a contact import definition where the sync action sets the record's status, where the instance ID is `9347bfe1-9c72-409c-a5cd-402ff74f0caa` and execution ID is `12345`:

```
POST https://secure.p03.eloqua.com/api/bulk/2.0/contacts/imports

{
  "name" : "AwesomeApp Decision Response Bulk Import",
  "updateRule" : "always",
```

```
  "fields" : {
    "emailAddress" : "{{Contact.Field(C_EmailAddress)}}"
  }
  "syncActions" : [
    {
      "destination" : "{{DecisionInstance
(9347bfe19c72409ca5cd402ff74f0caa).Execution[12345]}}",
      "action" : "setStatus",
      "status" : "yes"
    }
  ],
  "identifierFieldName" : "emailAddress"
}
```

Eloqua's response will be a 201 created response that includes a `uri` parameter, which you can use to identify the import:

```
{
  "name": "AwesomeApp Decision Response Bulk Import",
  "updateRule": "always",
  "fields": {
    "emailAddress": "{{Contact.Field(C_EmailAddress)}}"
  },
  "identifierFieldName": "emailAddress",
  "syncActions": [
    {
      "destination": "{{DecisionInstance(9347bfe19c72409ca5cd402ff74f0caa).Execution
[12345]}}",
```

```
    "action": "setStatus",
    "status": "yes"
   }
 ],
 "isSyncTriggeredOnImport": false,
 "isUpdatingMultipleMatchedRecords": false,
 "uri": "/contacts/imports/6",
 "createdBy": "DocsExample",
 "createdAt": "2014-03-06T13:59:00.6600046Z",
 "updatedBy": "DocsExample",
 "updatedAt": "2014-03-06T13:59:00.6600046Z"
}
```

2. Send Eloqua the import data using the `uri`:

```
POST https://secure.p03.eloqua.com/api/bulk/2.0/contacts/imports/6/data
```

```
[
   {
     "emailAddress" : "fred@example.com"
   },
   {
     "emailAddress" : "sylvie@example.com"
   }
]
```

3. Synchronize the data for import:

   AwesomeApp's request:

```
POST https://secure.p03.eloqua.com/api/bulk/2.0/syncs
```

```
{
   "syncedInstanceURI":"/contacts/imports/6"
}
```

Eloqua's response:

```
201 Created


{
    "syncedInstanceURI" : "/contacts/imports/6",
    "status" : "pending",
    "createdAt" : "2014-01-01T13:59:07.1375620Z",
    "createdBy" : "DocsExample",
    "uri" : "/syncs/6"
}
```

4. You can then use the sync's URI (`/syncs/6`) to check the status of the sync:

```
GET https://secure.p03.eloqua.com/api/bulk/2.0/sync/6
```

When the sync is complete, Eloqua's response will resemble:

```
200 OK


{
     "syncedInstanceURI" : "/contacts/imports/6",
     "syncStartedAt" : "2014-01-01T13:59:07.1375620Z",
     "syncEndedAt" : "2014-01-01T13:59:14.1375620Z",
     "status" : "success",
     "createdAt" : "2014-01-01T13:59:07.1375620Z",
```

```
    "createdBy" : "DocsExample",
    "uri" : "/syncs/6"
}
```

Bulk API custom object import

1. Create the bulk import definition, setting the status to **yes** to import data.

   If there is no data to import, and you only need to update a record's status, you can update a record's status without performing an import by creating a custom object sync action definition.

   When importing, the destination field refers to the decision service's instance - in this example, the instance GUID is `f82d50cd-86a9-4fca-b37e-4ec9a98b0339`:

   > **Warning**: When referencing service instances, you must transmit the GUID without dashes. The bulk API will error if you transmit the GUID with the dashes.

   Create a custom object import definition where the sync action sets the record's status, where the instance ID is `f82d50cd-86a9-4fca-b37e4ec9a98b0339` and execution ID is `12345`:

```
POST https://secure.p03.eloqua.com/api/bulk/2.0/customObjects/9/imports

{
  "name" : "AwesomeApp Decision Response Bulk Import",
  "updateRule" : "always",
  "fields" : {
    "email" : "{{CustomObject[9].Field[58]}}"
  },
  "syncActions" : [
```

```
    {
      "destination" : "{{DecisionInstance
(f82d50cd86a94fcab37e4ec9a98b0339).Execution[12345]}}",
      "action" : "setStatus",
      "status" : "yes"
    }
  ],
  "identifierFieldName" : "email"
}
```

Eloqua's response will be a 201 created response that includes a `uri` parameter, which you can use to identify the import:

```
HTTP/1.1 201 Created

{
    "name" : "AwesomeApp Decision Response Bulk Import",
    "updateRule" : "always",
    "fields" : {
      "email" : "{{CustomObject[9].Field[58]}}"
    },
    "identifierFieldName" : "email",
    "syncActions" : [
      {
        "destination" : "{{DecisionInstance
(f82d50cd86a94fcab37e4ec9a98b0339).Execution[12345]}}",
        "action" : "setStatus",
        "status" : "yes"
      }
    ],
```

```
    "isSyncTriggeredOnImport" : false,
    "isUpdatingMultipleMatchedRecords" : false,
    "uri" : "/customObjects/imports/9",
    "createdBy" : "DocsExample",
    "createdAt" : "2014-03-06T13:59:00.6600046Z",
    "updatedBy" : "DocsExample",
    "updatedAt" : "2014-03-06T13:59:00.6600046Z"
  }
```

2. Send Eloqua the data for import as a using the `uri`:

```
POST https://secure.p03.eloqua.com/api/bulk/2.0/customObjects/imports/9/data
```

```
[
   {
     "email" : "fred@example.com"
   },
   {
     "email" : "sylvie@example.com"
   }
]
```

3. Synchronize the data for import:

   AwesomeApp's request:

```
POST https://secure.p03.eloqua.com/api/bulk/2.0/syncs
```

```
{
   "syncedInstanceURI":"/customObjects/imports/9"
}
```

Eloqua's response:

```
201 Created


{
    "syncedInstanceURI" : "/customObjects/imports/9",
    "status" : "pending",
    "createdAt" : "2014-01-01T13:59:07.1375620Z",
    "createdBy" : "DocsExample",
    "uri" : "/syncs/9"
}
```

4. You can then use the sync's URI (`/syncs/9`) to check the status of the sync:

```
GET https://secure.p03.eloqua/api/bulk/2.0/sync/9
```

When the sync is complete, Eloqua’s response will resemble:

```
200 OK


{
     "syncedInstanceURI" : "/customObjects/imports/9",
     "syncStartedAt" : "2014-01-01T13:59:07.1375620Z",
     "syncEndedAt" : "2014-01-01T13:59:14.1375620Z",
     "status" : "success",
     "createdAt" : "2014-01-01T13:59:07.1375620Z",
```

```
    "createdBy" : "DocsExample",
    "uri" : "/syncs/9"
}
```

Learn more: Bulk API imports.

Delete URL

The delete URL is a templated URL pointing to an endpoint for deleting an instance of your service.

The delete URL uses an HTTP DELETE request and there is no content sent in the request body. All common URL template parameters are available (the same as with a create URL). On success, this endpoint should return a 200-level response.

An example delete URL would look something like:

```
https://www.someurl.com/delete/{appId}/{installId}/{instanceId}/{userName}/{siteName}/{siteId}
```

Note: Delete calls are not sent in real time, but are done in a batch once daily.

Develop an Oracle Eloqua app content service

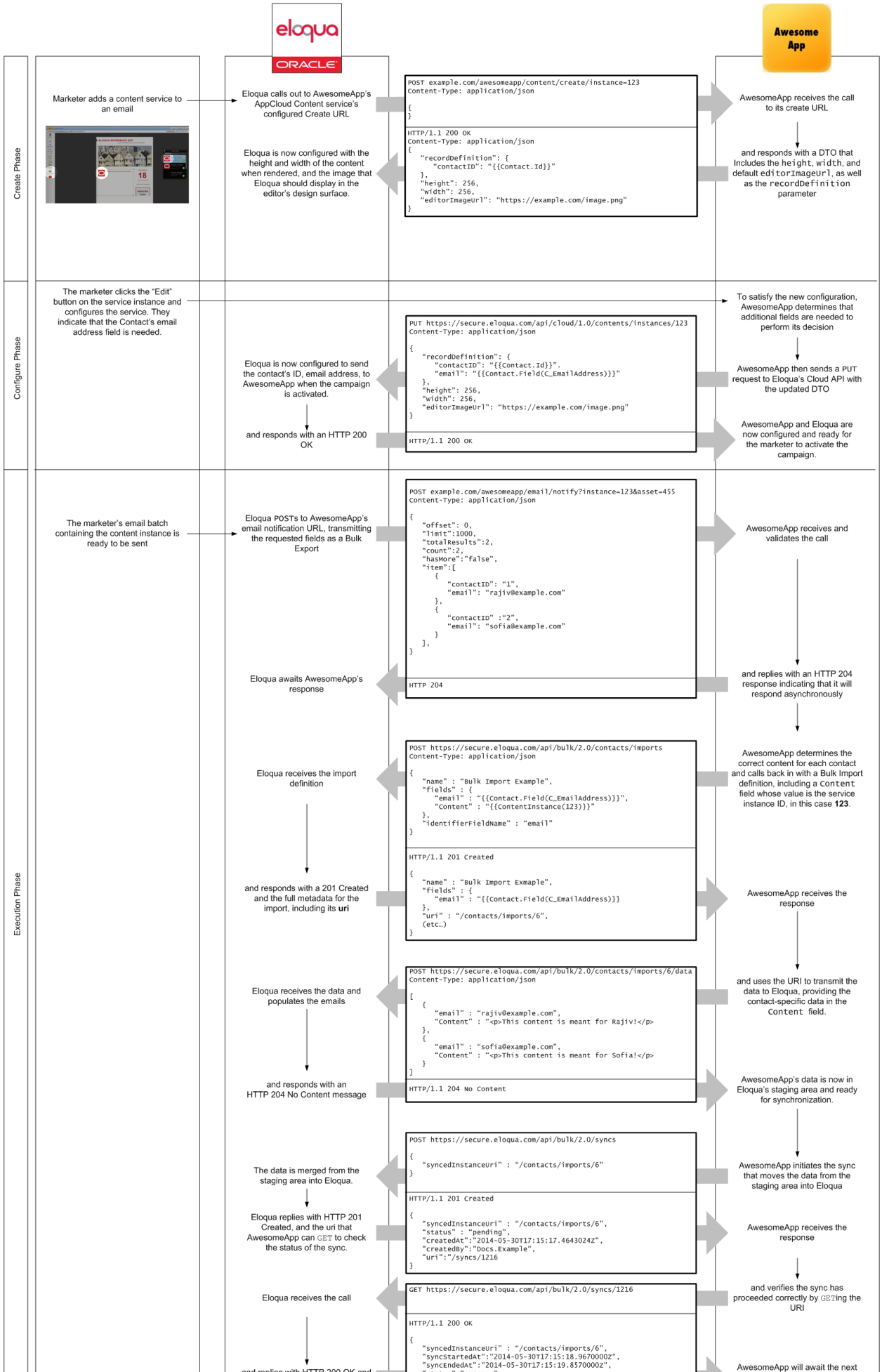
Content allows marketers to source pieces of content in Eloqua emails and landing pages, from an external source. This is the new and improved version of Eloqua's cloud components framework, and it includes many improvements such as asynchronous bulk processing (for emails), the ability to fully test the content service within Eloqua, and design-time interaction with the email and landing page editors.

Content follows the instantiation-execution model.The following sections describe content-specific details and provide the endpoints needed to develop a content service. If you are not familiar with the general flow for the instantiation-execution model, you should read that first.

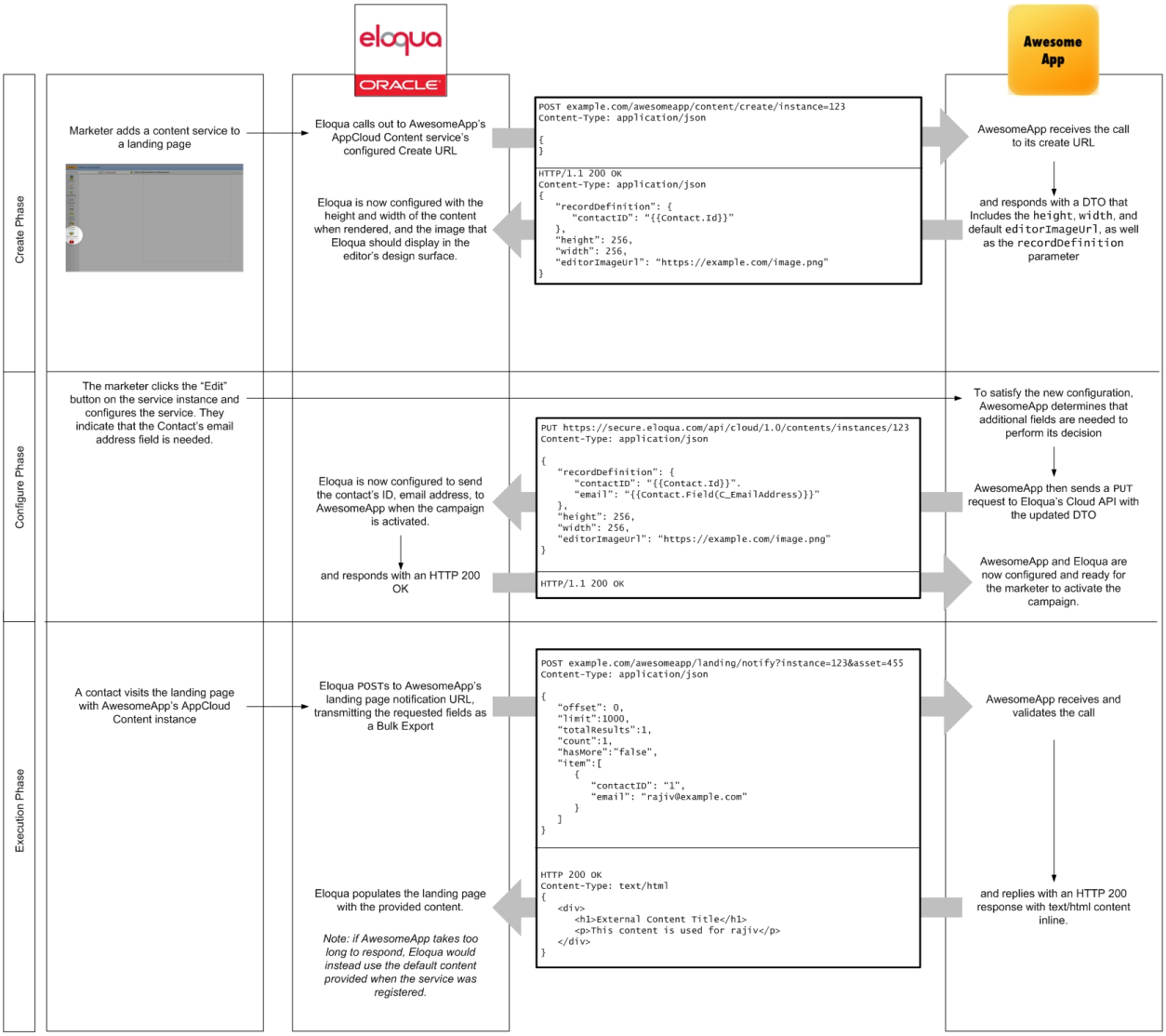
Eloqua signs all outgoing calls with OAuth 1.0a so the receiving system can validate that the call was sent by Eloqua. Refer to the OAuth 1.0a spec or OAuth 1.0 RFC for more information.

Content service flow diagrams

For emails


eloqua
ORACLE
Awesome App
Create Phase
Marketer adds a content service to an email
Eloqua calls out to AwesomeApp's AppCloud Content service's configured Create URL
POST example.com/awesomeapp/content/create/instance=123
Content-Type: application/json
{
}
AwesomeApp receives the call to its create URL
HTTP/1.1 200 OK
Content-Type: application/json
{
"recordDefinition": {
"contactID": "{{Contact.Id}}"
},
"height": 256,
"width": 256,
"editorImageUrl": "https://example.com/image.png"
}
Eloqua is now configured with the height and width of the content when rendered, and the image that Eloqua should display in the editor's design surface.
and responds with a DTO that Includes the height, width, and default editorImageUrl, as well as the recordDefinition parameter
Configure Phase
The marketer clicks the "Edit" button on the service instance and configures the service. They indicate that the Contact's email address field is needed.
To satisfy the new configuration, AwesomeApp determines that additional fields are needed to perform its decision
PUT https://secure.eloqua.com/api/cloud/1.0/contents/instances/123
Content-Type: application/json
{
"recordDefinition": {
"contactID": "{{Contact.Id}}".
"email": "{{Contact.Field(C_EmailAddress)}}"
},
"height": 256,
"width": 256,
"editorImageUrl": "https://example.com/image.png"
}
Eloqua is now configured to send the contact's ID, email address, to AwesomeApp when the campaign is activated.
AwesomeApp then sends a PUT request to Eloqua's Cloud API with the updated DTO
HTTP/1.1 200 OK
and responds with an HTTP 200 OK
AwesomeApp and Eloqua are now configured and ready for the marketer to activate the campaign.
Execution Phase
The marketer's email batch containing the content instance is ready to be sent
Eloqua POSTs to AwesomeApp's email notification URL, transmitting the requested fields as a Bulk Export
POST example.com/awesomeapp/email/notify?instance=123&asset=455
Content-Type: application/json
{
"offset": 0,
"limit":1000,
"totalResults":2,
"count":2,
"hasMore":"false",
"item":[
{
"contactID": "1",
"email": "rajiv@example.com"
},
{
"contactID" :"2",
"email": "sofia@example.com"
}
],
}
AwesomeApp receives and validates the call
HTTP 204
Eloqua awaits AwesomeApp's response
and replies with an HTTP 204 response indicating that it will respond asynchronously
POST https://secure.eloqua.com/api/bulk/2.0/contacts/imports
Content-Type: application/json
{
"name" : "Bulk Import Example",
"fields" : {
"email" : "{{Contact.Field(C_EmailAddress)}}",
"Content" : "{{ContentInstance(123)}}"
},
"identifierFieldName" : "email"
}
AwesomeApp determines the correct content for each contact and calls back in with a Bulk Import definition, including a Content field whose value is the service instance ID, in this case 123.
Eloqua receives the import definition
HTTP/1.1 201 Created
{
"name" : "Bulk Import Exmaple",
"fields" : {
"email" : "{{Contact.Field(C_EmailAddress)}}
},
"uri" : "/contacts/imports/6",
(etc…)
}
and responds with a 201 Created and the full metadata for the import, including its uri
AwesomeApp receives the response
POST https://secure.eloqua.com/api/bulk/2.0/contacts/imports/6/data
Content-Type: application/json
[
{
"email" : "rajiv@example.com",
"Content" : " This content is meant for Rajiv!
},
{
"email" : "sofia@example.com",
"Content" : " This content is meant for Sofia!
}
]
and uses the URI to transmit the data to Eloqua, providing the contact-specific data in the Content field.
Eloqua receives the data and populates the emails
HTTP/1.1 204 No Content
and responds with an HTTP 204 No Content message
AwesomeApp's data is now in Eloqua's staging area and ready for synchronization.
POST https://secure.eloqua.com/api/bulk/2.0/syncs
{
"syncedInstanceUri" : "/contacts/imports/6"
}
AwesomeApp initiates the sync that moves the data from the staging area into Eloqua
The data is merged from the staging area into Eloqua.
HTTP/1.1 201 Created
{
"syncedInstanceUri" : "/contacts/imports/6",
"status" : "pending",
"createdAt":"2014-05-30T17:15:17.4643024Z",
"createdBy":"Docs.Example",
"uri":"/syncs/1216
}
Eloqua replies with HTTP 201 Created, and the uri that AwesomeApp can GET to check the status of the sync.
AwesomeApp receives the response
GET https://secure.eloqua.com/api/bulk/2.0/syncs/1216
and verifies the sync has proceeded correctly by GETing the URI
Eloqua receives the call
HTTP/1.1 200 OK
{
"syncedInstanceUri" : "/contacts/imports/6",
"syncStartedAt":"2014-05-30T17:15:18.9670000Z",
"syncEndedAt":"2014-05-30T17:15:19.8570000Z",
AwesomeApp will await the next

For landing pages



Create URL

Eloqua's call to the **CreateURL**:

```
POST https://example.com/awesomeapp/content/create?instance=f82d50cd-86a9-4fca-b37e-4ec9a98b0339

{}
```

For a content service, AwesomeApp's response must include `height`, `width`, and default `editorImageUrl`, as well as the `recordDefinition` parameter. You can include a maximum of 250 fields in your record definition.

The `height` and `width` parameters define the size of the content instance when rendered, while `editorImageUrl` specifies the URL for an image that Eloqua will display in the editor's design surface. `editorImageUrl` is **not** a templated URL.

> **Note**: The "Content-Type" header must be set to "application/json" for the response to the Create URL call.

```
HTTP/1.1 200 application/json; charset=utf-8

{
  "recordDefinition": {
    "ContactID": "{{Contact.Id}}"
  },
  "height": 256,
  "width": 256,
  "editorImageUrl": "https://example.com/image.png",
  "requiresConfiguration": true
}
```

`requiresConfiguration` is an optional Boolean parameter which tells Eloqua whether user configuration is required before the app can be used. If set to `true`, users will be unable to save an email or landing page asset containing the unconfigured app service instance. Eloqua will display an error message.

> **Warning**: All field names and values in Eloqua data transfer objects are case sensitive. Be sure to reproduce them exactly. For example: `{{Contact.id}}` would fail if the correct form is `{{Contact.Id}}`.

Configure URL

When a user clicks the edit button on the service instance and makes changes through the **Configure URL** that require an updated recordDefinition DTO, AwesomeApp must

call out to Eloqua's cloud API `PUT /contents/instances/{id}` endpoint with that updated DTO:

```
PUT https://secure.eloqua.com/api/cloud/1.0/contents/instances/f82d50cd-86a9-4fca-b37e-4ec9a98b0339

{
  "recordDefinition": {
    "ContactID": "{{Contact.Id}}",
    "EmailAddress": "{{Contact.Field(C_EmailAddress)}}",
  },
  "height": 256,
  "width": 256,
  "editorImageUrl": "https://example.com/image.png",
  "requiresConfiguration": false
}
```

`requiresConfiguration` is an optional Boolean parameter which tells Eloqua whether user configuration is required before the app can be used. If set to `true`, users will be unable to save an email or landing page asset containing the unconfigured app service instance. Eloqua will display an error message.

Use the configure URL response to set `requiresConfiguration` to `false` when your app's configuration acceptance criteria have been met.

> **Note**: `X-Frame-Options` must not be set to `DENY` for any configure page.

> **Warning**: If the email or landing page asset is not in draft mode, attempting to set `requiresConfiguration` to `true` will result in an error.

Notification URL

Eloqua's call to the **Notification URL** transmits the requested fields in the `items` parameter:

```
POST https://example.com/awesomeapp/content/notify?instance=f82d50cd-86a9-4fca-b37e-4ec9a98b0339&asset=456

{
   "offset": 0,
   "limit": 1000,
   "totalResults": 2,
   "count": 2,
   "hasMore": false,
   "items":
   [
     {
       "ContactID": "1",
       "EmailAddress": "fred@example.com"
     },
     {
       "ContactID": "2",
       "EmailAddress": "john@example.com"
     }
   ]
}
```

For content services, AwesomeApp's response will depend on whether the content is for an email or a landing page:

For landing pages, the response is a 200 status code with text/html content inline. If the response takes too long, Eloqua uses the default content for that contact.

For **Email**, AwesomeApp can respond in one of two ways:

- *Inline response*: A 200 status code with text/html content inline, in which case Eloqua uses the same content for each contact (this is the same response type as used for landing pages), **or**
- *Asynchronous response*: Asynchronous response: A 204 status code, indicating that the call was accepted, but there is no content to return directly. The service should process asynchronously and then call the bulk API. Eloqua waits 24 hours for a response. If there is no response from the app after 24 hours the email is not sent and those records are set to error.

Important: If there is no response or an error returned for the Notification Call Eloqua uses the default content for that contact.

Inline response

The inline response for landing pages and email is a 200 status code, followed by the text/html content. For example:

```
{
  <div>
    <h1>External Content</h1>
    <p>This content is used for all recipients for this request.</p>
  </div>
}
```

Asynchronous response

For the asynchronous response, AwesomeApp responds with a 204 status code, indicating that the response will be asynchronous, followed by one or more imports to Eloqua's bulk API, where contact-specific content is updated with the ContactId and the instance Id, mapping the contact to the new html content.

Note: The maximum import size per batch is 5,000. If you have more than 5,000 contacts, break your data up into multiple imports.

1. Create the bulk import definition, including a `Content` parameter whose value is the service instance ID and execution ID. Including the execution ID enables you to differentiate between different uses of the asset, for example, when multiple campaigns use the same email template that contains the service instance:

   ```
   POST https://secure.eloqua.com/api/bulk/2.0/contacts/imports
   {
   ```

```
  "name": "AwesomeApp Content Response Bulk Import",
  "updateRule": "always",
  "fields": {
    "EmailAddress": "{{Contact.Field(C_EmailAddress)}}",
    "Content": "{{ContentInstance(f82d50cd86a94fcab37e4ec9a98b0339).Execution
[21]}}"
  },
  "identifierFieldName": "EmailAddress"
}
```

Eloqua's response will be a 201 Created response that includes a `uri` parameter, which you can use to identify the import:

```
HTTP/1.1 201 Created
{
  "name" : "AwesomeApp Content Response Bulk Import",
  "updateRule" : "always",
  "fields" : {
    "EmailAddress" : "{{Contact.Field(C_EmailAddress)}}",
    "Content" : "{{ContentInstance(f82d50cd86a94fcab37e4ec9a98b0339).Execution
[21]}}"
  },
  "identifierFieldName" : "EmailAddress",
  "isSyncTriggeredOnImport" : false,
  "isUpdatingMultipleMatchedRecords" : false,
  "uri" : "/contacts/imports/6",
  "createdBy" : "DocsExample",
  "createdAt" : "2014-03-06T13:59:00.6600046Z",
  "updatedBy" : "DocsExample",
```

```
  "updatedAt" : "2014-03-06T13:59:00.6600046Z"
}
```

2. Send Eloqua the data for import, using the content parameter with the content the Eloqua should use for each contact:

```
POST https://secure.eloqua.com/api/bulk/2.0/contacts/imports/6/data
[
  {
    "EmailAddress" : "fred@example.com",
    "Content" : "<p>This is the content for Fred</p>"
  },
  {
    "EmailAddress" : "sylvie@example.com",
    "Content" : "<p>This is the content for Sylvie</p>"
  }
]
```

3. Synchronize the data for import:

   AwesomeApp's request:

```
POST https://secure.p03.eloqua.com/api/bulk/2.0/syncs

{
   "syncedInstanceURI":"/contacts/imports/6"
}
```

   Eloqua's response:

```
201 Created
```

```
{
    "syncedInstanceURI" : "/contacts/imports/6",
    "status" : "pending",
    "createdAt" : "2014-01-01T13:59:07.1375620Z",
    "createdBy" : "DocsExample",
    "uri" : "/syncs/6"
}
```

4. You can then use the sync's URI (`/syncs/6`) to check the status of the sync:

```
GET https://secure.p03.eloqua.com/api/bulk/2.0/sync/6
```

When the sync is complete, Eloqua's response will resemble:

```
200 OK


{
    "syncedInstanceURI" : "/contacts/imports/6",
    "syncStartedAt" : "2014-01-01T13:59:07.1375620Z",
    "syncEndedAt" : "2014-01-01T13:59:14.1375620Z",
    "status" : "success",
    "createdAt" : "2014-01-01T13:59:07.1375620Z",
    "createdBy" : "DocsExample",
    "uri" : "/syncs/6"
}
```

Notes:

- If you want your content service hyperlinks to take advantage of Eloqua's dynamic link tracking features, it will not do so automatically. To enable Eloqua tracking for a link within your content service:
  - If the destination page has an Eloqua tracking script installed you should append `s=<span class=eloquaemail>siteid</span>&elq=<span class=eloquaemail>recipientid</span>` to the URL.
  - If the destination page does not have an Eloqua tracking script installed you should append `elqtrack=true` to the URL.
- The http protocol must be included to use Eloqua tracking parameters. For example:
  - `google.com?elqTrack=true` - Detected as invalid URL. No clickthrough is recorded.
  - `http://google.com?elqTrack=true` - Detected as valid URL. Clickthrough is recorded.

Delete URL

The delete URL is a templated URL pointing to an endpoint for deleting an instance of your service.

The delete URL uses an HTTP DELETE request and there is no content sent in the request body. All common URL template parameters are available (the same as with a create URL). On success, this endpoint should return a 200-level response.

An example delete URL would look something like:

```
https://www.someurl.com/delete/{appId}/{installId}/{instanceId}/{userName}/{siteName}/{siteId}
```

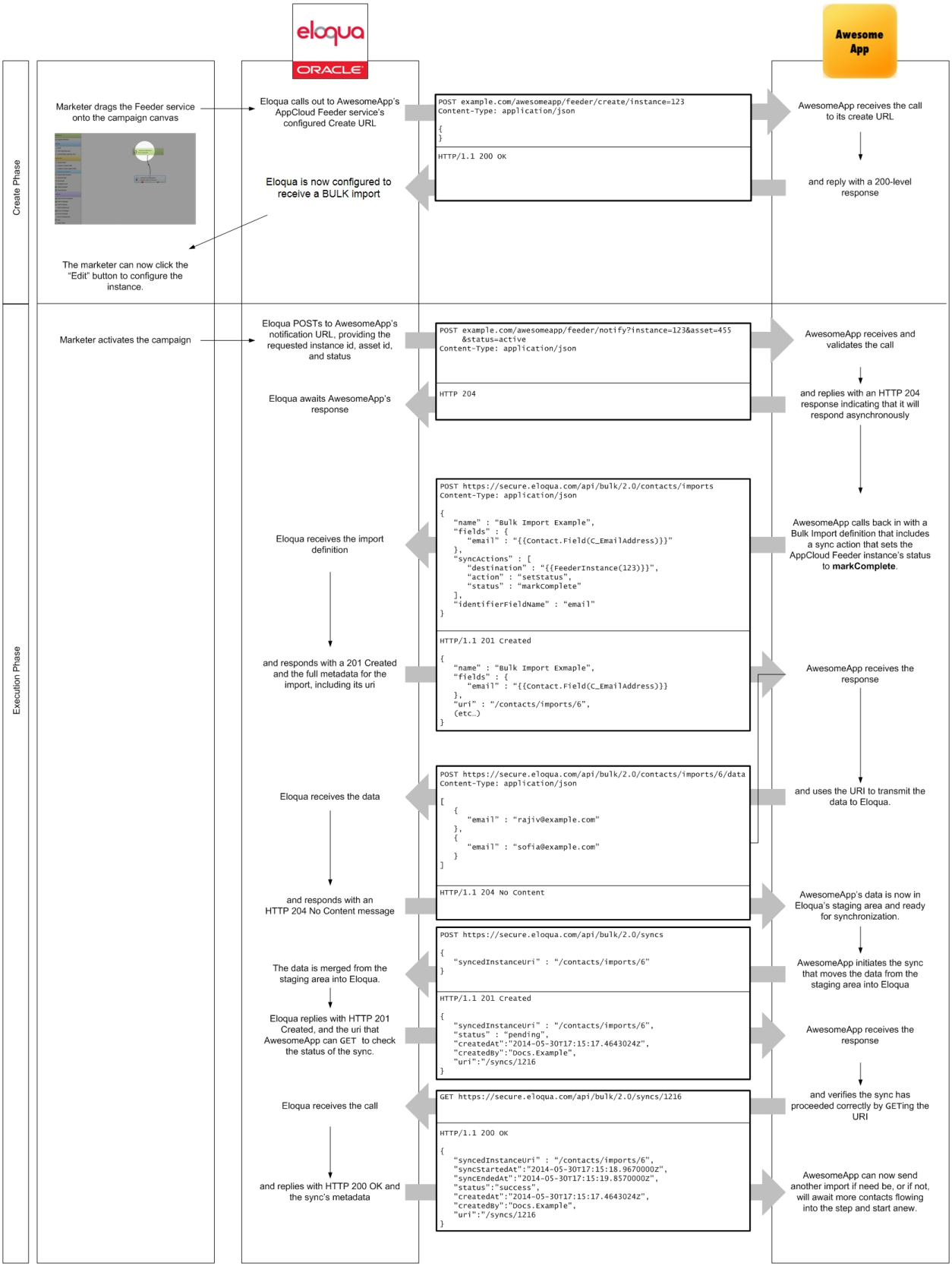
Note: Delete calls are not sent in real time, but are done in a batch once daily.

Develop an Oracle Eloqua app feeder service

Feeders allow external systems to determine and control which contacts or custom objects enter into a campaign or program canvas, and when. Developers can now build apps that utilize data in third-party systems to drive campaign or program membership.

Feeders follow the instantiation-execution model. The following sections describe feeder-specific details and provide the endpoints needed to develop a feeder service. If you are not familiar with the general flow for the instantiation-execution model, you should read that first.

Eloqua signs all outgoing calls with OAuth 1.0a so the receiving system can validate that the call was sent by Eloqua. Refer to the OAuth 1.0a spec or OAuth 1.0 RFC for more information.



Create URL

Eloqua's call to the **CreateURL**:

```
POST https://example.com/awesomeapp/feeders/create?siteid=123instance=9347bfe1-9c72-409c-a5cd-402ff74f0caa

{}
```

AwesomeApp's response is a 200-level HTTP response:

> **Note**: The "Content-Type" header must be set to "application/json" for the response to the Create URL call.

```
HTTP/1.1 200 application/json; charset=utf-8

{
  "requiresConfiguration": true
}
```

`requiresConfiguration` is an optional Boolean parameter which tells Eloqua whether user configuration is required before the app can be used. If set to `true`, users will be unable to activate a campaign or program containing the unconfigured app service instance. Eloqua will display an error message.

Configure URL

When a user clicks the edit button on the service instance and makes changes through the **Configure URL**, AwesomeApp must call out to Eloqua's Cloud API `PUT /feeders/instances/{id}` endpoint:

```
PUT https://secure.eloqua.com/api/cloud/1.0/feeders/instances/9347bfe1-9c72-409c-a5cd-402ff74f0caa
```

```
{
  "requiresConfiguration": false
}
```

`requiresConfiguration` is an optional Boolean parameter which tells Eloqua whether user configuration is required before the app can be used. If set to `true`, users will be unable to activate a campaign or program containing the unconfigured app service instance. Eloqua will display an error message.

Use the Configure URL response to set `requiresConfiguration` to `false` when your app's configuration acceptance criteria have been met.

> **Note**: `X-Frame-Options` must not be set to `DENY` for any configure page.

> **Warning**: If the campaign or program is not in draft mode, attempting to set `requiresConfiguration` to `true` will result in an error.

Notification URL

Eloqua's call to the **Campaign Status Notification URL** tells AwesomeApp that the campaign or program is active and ready to receive contacts or custom object records. These calls are made using templated URLs to differentiate the call out. For example, the notification call for activating a campaign would resemble:

```
POST https://example.com/awesomeapp/feeders/notify?instance=9347bfe1-9c72-409c-a5cd-402ff74f0caa&asset=456&status=Activated
```

When the campaign or program is deactivated, Eloqua's call to the **Campaign Status Notification URL** tells AwesomeApp to stop sending contacts or custom object records. The deactivate notification would resemble:

```
POST https://example.com/awesomeapp/feeders/notify?instance=9347bfe1-9c72-409c-a5cd-402ff74f0caa&asset=456&status=Draft
```

After the campaign or program is activated, AwesomeApp responds with a 204 response, indicating that the response will be asynchronous, followed by an **import** to Eloqua's Bulk API, where AwesomeApp specifies a sync action. For AppCloud Feeders, the sync action sets the AppCloud Feeder instance's status to **complete**. Note that the maximum import size per batch is 5,000: if you have more than 5,000 records, break your data up into multiple batches.

Bulk API contact import

1. Create the bulk import definition, setting the status of the AppCloud Feeder instance to **complete**. The "destination" field refers to the AppCloud Feeders service's instance - in this example, the instance ID is `9347bfe1-9c72-409c-a5cd-402ff74f0caa`:

   > **Warning**: When referencing service instances, you must transmit the GUID without dashes. The bulk API will error if you transmit the GUID with the dashes.

   Create a contact import definition where the sync action sets the record's status, where the instance ID is `9347bfe1-9c72-409c-a5cd-402ff74f0caa` :

   ```
   POST https://secure.p03.eloqua.com/api/bulk/2.0/contacts/imports
   {
     "name" : "AwesomeApp Feeder Bulk Import",
     "updateRule" : "always",
     "fields" : {
       "emailAddress" : "{{Contact.Field(C_EmailAddress)}}"
     }
   ```

```
  "syncActions" : [
    {
      "destination" : "{{FeederInstance(9347bfe19c72409ca5cd402ff74f0caa)}}",
      "action" : "setStatus",
      "status" : "complete"
    }
  ],
  "identifierFieldName" : "emailAddress"
}
```

> **Warning**: All field names and values in Eloqua data transfer objects are case sensitive. Be sure to reproduce them exactly. For example: `{{Contact.id}}` would fail if the correct form is `{{Contact.Id}}`.

Eloqua's response will be a 201 Created response that includes a `uri` parameter, which you can use to identify the import:

```
{
  "name": "AwesomeApp Feeder Bulk Import",
  "updateRule": "always",
  "fields": {
    "emailAddress": "{{Contact.Field(C_EmailAddress)}}"
  },
  "identifierFieldName": "emailAddress",
  "syncActions": [
```

```
    {
      "destination": "{{FeederInstance(9347bfe19c72409ca5cd402ff74f0caa)}}",
      "action": "setStatus",
      "status": "complete"
    }
  ],
  "isSyncTriggeredOnImport": false,
  "isUpdatingMultipleMatchedRecords": false,
  "uri": "/contacts/imports/6",
  "createdBy": "DocsExample",
  "createdAt": "2014-03-06T13:59:00.6600046Z",
  "updatedBy": "DocsExample",
  "updatedAt": "2014-03-06T13:59:00.6600046Z"
}
```

2. Send Eloqua the import data using the `uri`:

```
POST https://secure.p03.eloqua.com/api/bulk/2.0/contacts/imports/6/data
```

```
[
  {
    "emailAddress" : "fred@example.com"
  },
  {
    "emailAddress" : "sylvie@example.com"
  }
]
```

3. Synchronize the data for import:

   AwesomeApp's request:

```
POST https://secure.p03.eloqua.com/api/bulk/2.0/syncs

{
   "syncedInstanceURI":"/contacts/imports/6"
}
```

Eloqua's response:

```
201 Created


{
    "syncedInstanceURI" : "/contacts/imports/6",
    "status" : "pending",
    "createdAt" : "2014-01-01T13:59:07.1375620Z",
    "createdBy" : "DocsExample",
    "uri" : "/syncs/6"
}
```

4. You can then use the sync's URI (`/syncs/6`) to check the status of the sync:

```
GET https://secure.p03.eloqua.com/api/bulk/2.0/sync/6
```

When the sync is complete, Eloqua's response will resemble:

```
200 OK


{
     "syncedInstanceURI" : "/contacts/imports/6",
     "syncStartedAt" : "2014-01-01T13:59:07.1375620Z",
     "syncEndedAt" : "2014-01-01T13:59:14.1375620Z",
     "status" : "success",
```

```
    "createdAt" : "2014-01-01T13:59:07.1375620Z",
    "createdBy" : "DocsExample",
    "uri" : "/syncs/6"
}
```

Bulk API custom object import

1. Create the bulk import definition, setting the status to yes. The destination field refers to the feeder service's instance - in this example, the instance GUID is `f82d50cd-86a9-4fca-b37e-4ec9a98b0339`:

> **Warning**: When referencing service instances, you must transmit the GUID without dashes. The bulk API will error if you transmit the GUID with the dashes.

Create a custom object import definition where the sync action sets the record's status, where the instance ID is `f82d50cd-86a9-4fca-b37e4ec9a98b0339` :

```
POST https://secure.p03.eloqua.com/api/bulk/2.0/customObjects/9/imports

{
  "name" : "AwesomeApp Feeder Response Bulk Import",
  "updateRule" : "always",
  "fields" : {
    "email" : "{{CustomObject[9].Field[58]}}"
  },
  "syncActions" : [
    {
      "destination" : "{{FeederInstance(f82d50cd86a94fcab37e4ec9a98b0339)}}",
      "action" : "setStatus",
```

```
      "status" : "complete"
    }
  ],
  "identifierFieldName" : "email"
}
```

Eloqua's response will be a 201 Created response that includes a `uri` parameter, which you can use to identify the import:

```
HTTP/1.1 201 Created

{
    "name" : "AwesomeApp Feeder Response Bulk Import",
    "updateRule" : "always",
    "fields" : {
      "email" : "{{CustomObject[9].Field[58]}}"
    },
    "identifierFieldName" : "email",
    "syncActions" : [
      {
        "destination" : "{{FeederInstance(f82d50cd86a94fcab37e4ec9a98b0339)}}",
        "action" : "setStatus",
        "status" : "complete"
      }
    ],
    "isSyncTriggeredOnImport" : false,
    "isUpdatingMultipleMatchedRecords" : false,
    "uri" : "/customObjects/imports/9",
    "createdBy" : "DocsExample",
    "createdAt" : "2014-03-06T13:59:00.6600046Z",
```

```
    "updatedBy" : "DocsExample",
    "updatedAt" : "2014-03-06T13:59:00.6600046Z"
  }
```

2. Send Eloqua the data for import as a using the `uri`:

```
POST https://secure.p03.eloqua.com/api/bulk/2.0/customObjects/imports/9/data
```

```
[
   {
     "email" : "fred@example.com"
   },
   {
     "email" : "sylvie@example.com"
   }
]
```

3. Synchronize the data for import:

   AwesomeApp's request:

```
POST https://secure.p03.eloqua.com/api/bulk/2.0/syncs

{
   "syncedInstanceURI":"/customObjects/imports/9"
}
```

   Eloqua's Response:

```
201 Created


{
    "syncedInstanceURI" : "/customObjects/imports/9",
    "status" : "pending",
    "createdAt" : "2014-01-01T13:59:07.1375620Z",
    "createdBy" : "DocsExample",
    "uri" : "/syncs/9"
}
```

4. You can then use the sync's uri (`/syncs/9`) to check the status of the sync:

```
GET https://secure.p03.eloqua.com/api/bulk/2.0/sync/9
```

When the sync is complete, Eloqua's response will resemble:

```
200 OK


{
    "syncedInstanceURI" : "/customObjects/imports/9",
    "syncStartedAt" : "2014-01-01T13:59:07.1375620Z",
    "syncEndedAt" : "2014-01-01T13:59:14.1375620Z",
    "status" : "success",
    "createdAt" : "2014-01-01T13:59:07.1375620Z",
    "createdBy" : "DocsExample",
    "uri" : "/syncs/9"
}
```

Learn more: Bulk API imports.

Delete URL

The delete URL is a templated URL pointing to an endpoint for deleting an instance of your service.

The delete URL uses an HTTP DELETE request and there is no content sent in the request body. All common URL template parameters are available (the same as with a create URL). On success, this endpoint should return a 200-level response.

An example delete URL would look something like:

```
https://www.someurl.com/delete/{appId}/{installId}/{instanceId}/{userName}/{siteName}/{siteId}
```

> **Note**: Delete calls are not sent in real time, but are done in a batch once daily.

Develop an Oracle Eloqua app menu service

The menus service is all about context. With menus, you can build an App that enables a marketer to launch an externally-hosted application from within Eloqua via a menu dock. This menu dock floats on the right side of the screen in the Eloqua's Campaign Canvas or the asset editor screens. When a marketer clicks on your menu app, Eloqua calls out to your service via the action URL, passing along context-specific data to your app.

Suppose AwesomeApp has a menu service. When a marketing user clicks on AwesomeApp, they are directed to the callout URL: `https://example.com/awesome/callout`. The call can include contextual information for your app to use. Available attributes include:

- `siteId` and `siteName`: these describe the Eloqua "site", generally the company or organization of the user
- `userId` and `userName`: the Eloqua username and id for the user who selected the menu service

- `userCulture`: the linguistic profile of the user. Eloqua uses Microsoft's CultureInfo class to define culture. For example, for English (United States), the code is `en-US`; for French (Canada), `fr-CA`
- `assetId` and `assetName`: the ID and name of the asset that the user was viewing when they opened the menu
- `assetType`: the type of asset, that the user was viewing when they opened the menu. Assets include account, campaign, contact, landing page, template, and so on.

The actual call to AwesomeApp would then resemble:

```
GET
https://example.com/awesome/callout?siteId=123&siteName=Eloqua&userId=1234&userName=Docs.Example&userCulture=en-US&assetId=4&assetName=Docs%20Example%20Campaign&assetType=Campaign
```

Eloqua signs all outgoing calls with OAuth 1.0a so the receiving system can validate that the call was sent by Eloqua. Refer to the OAuth 1.0a spec or OAuth 1.0 RFC for more information.

Develop an Oracle Eloqua app firehose service

Eloqua's firehose service type enables third-parties to build a service that receives a notification when a marketer performs an action in Eloqua. When a marketer creates a campaign or edits an email, for example, Eloqua can call out to the third-party with the update.

Firehose supports subscriptions to a wide variety of events for the different asset types:

- **Email**: Created, Updated, Deleted
- **LandingPage**: Created, Updated, Deleted
- **ContactSegment**: Created, Updated, Deleted
- **Form**: Created, Updated, Deleted

- **Campaign**: Created, Updated, Deleted, Draft, DraftApproved, DraftWaitingApproval, ActivatedBySchedule, CampaignLandingPageAdded, CampaignEmailAdded, CampaignFormAdded, CampaignLandingPageRemoved, CampaignEmailRemoved
- **Program**: Created, Updated, Deleted, Draft, Activated, Paused

When you configure your app, you specify which events to subscribe to in the pattern. Then, when a marketer triggers one of those events, Eloqua calls out to the configured **notification URL** with the data specified in the firehose pattern.

For example, AwesomeApp has a firehose service and is configured to request all campaign-related events. The call to its notification URL would then resemble:

```
POST https://example.com/firehose
{
   "siteId": "123",
   "assetId": "9999",
   "assetType": "Campaign",
   "eventDate": "12/12/2014 12:00:01 AM",
   "eventType": "Updated",
   "userId": "1234",
   "userName": "Docs.Example",
   "msgAttributes": {
      "name": "name_of_updated_asset",
      "statuschangedby": "1234",
      "statuschangedat": "12/12/2014 12:00:00 AM"
   },
   "oath_consumerkey": "stuff"
}
```

Eloqua signs all outgoing calls with OAuth 1.0a so the receiving system can validate that the call was sent by Eloqua. Refer to the OAuth 1.0a spec or OAuth 1.0 RFC for more information.

Note: If a firehose service is configured to receive form-related or segment-related events, app providers should expect to see two events whenever a form or segment is created or saved. When a form is created, Eloqua sends a `POST` request (for the

creation) and a `PATCH` request (for the update). App providers should expect to see two events in these scenarios.

Register your app

When you have developed an app and are ready to start testing it, you need to register it with Eloqua. By now, you will have already registered as an app provider in your Eloqua development instance. Now, you need to register your app and services.

Once registered, you can test your apps within your instance to make sure they work as expected, and release the apps selectively or to the general public.

Step 1: Register the app

To register your app, select **Settings** > **AppCloud Developer**.

If you have not yet registered any apps, you will see a “You currently have no apps” message, and the **Create App** button.

If you have already registered an app, it will be listed alongside the **Create App** button. To register your app, click **Create App**.

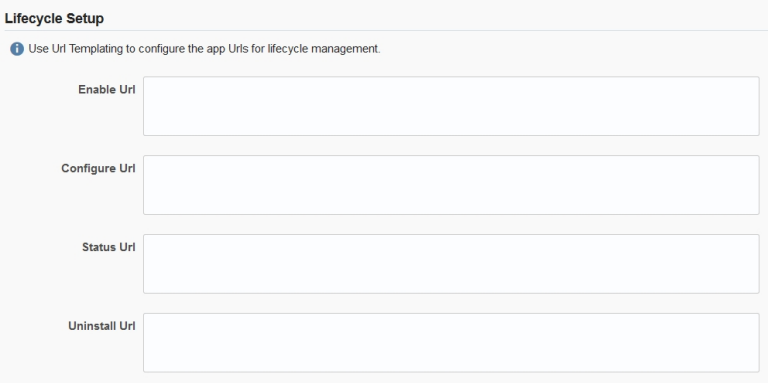
App information

The *Create App* screen prompts you to fill in a number of fields that describe your app. Complete these fields with the relevant information. Current fields include **Name**, **Description**, and **Icon**.

❑**Warning**: the **Name** field is limited to 100 characters and the **Description** field is limited to 4000 characters.

Lifecycle fields

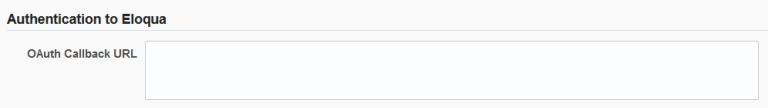
- **Enable URL**:The URL called when the App is first installed or when it is reconfigured. This is a templated URL which should include, at a minimum, the {InstallId}, {AppId}, and {CallbackUrl} parameters. WIthout the callback URL you will not be able to complete the installation flow if your app requires configuring (for example, to use OAuth).
- **Configure URL**: The URL called when the app is configured.
- **Status URL**:The URL called when a user checks the App's connection. Learn how to respond when Eloqua calls the Status URL.
- **Uninstall URL**: The URL called when a user uninstalls the app. This allows Eloqua to notify the app provider when someone uninstalls their app from the appcloud catalog. This URL supports templated parameters such as {InstallId}, {AppId}, etc.



Authentication to Eloqua

Apps for the AppCloud use OAuth for authentication.

- **OAuth Callback URL**: the URL that the users should be redirected to after installing the app and authenticating with Eloqua.



To use OAuth, you'll need the following information which is displayed after you click **Save**:

- **Client ID (app ID)**: Eloqua creates and assigns the app ID when you save your app.
- **Client Secret**: Eloqua generates the secret access token when you save your app. Click Show to view it.

Once you complete all the fields, click **Save** to finish configuring the app, or click **Add Services** to start registering services.

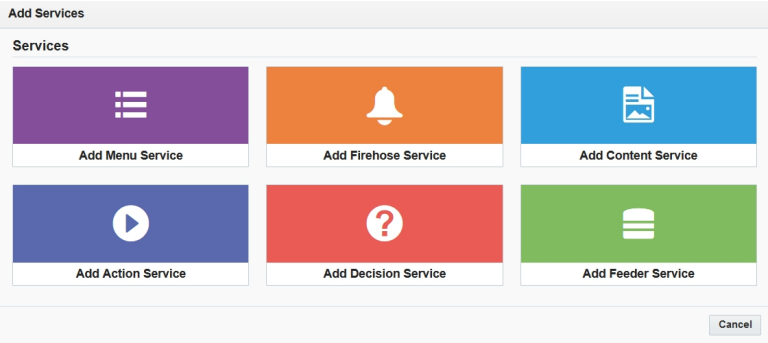
Step 2: Register services

Finally, you should register any service your app uses. The AppCloud developer platform currently supports 6 services: menus, firehose, content, actions, decisions, and feeders.

Apps can include zero services, as in the case of a portal app or if the integration not directly tied to one of the service types, or if it is tied to many services.

To define your app's services within Eloqua, navigate to **Settings** > **AppCloud Developer** then click on your app (see step 1 above for setup instructions) and select **Add Services**.

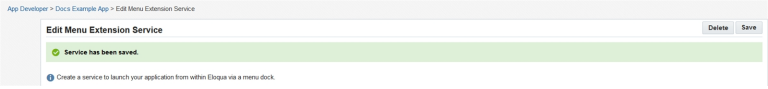
A modal opens listing the available service types. Select the service type you wish to register.



Then, fill out the service's details: you will have different information you need to supply depending on the service you are implementing. For example, for a menu, you need to provide the action URL, specify the areas it supports and how Eloqua should present it, and provide the service's icon URL, name, and description; for content, you will need to fill out instance configuration details, as well as configure content and notification settings, as well as the general service details. The following documents describe the service configuration page for each service type:

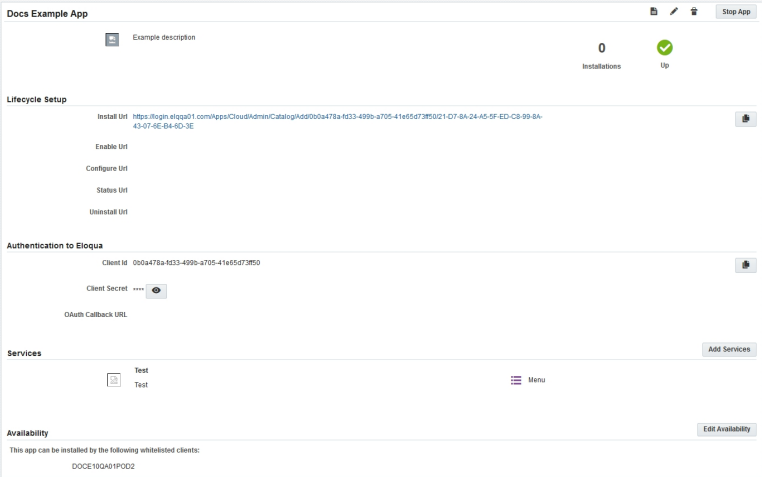
- Menus
- Content
- Actions
- Decisions
- Feeders
- Firehose

When you are done configuring your service, click **Save**. A green alert message will appear at the top of the page indicating the service has been successfully saved.



Step 3: View the app

You can view the app information by clicking its name in the list. Here is an example of the *App Details* page for *Docs Example App*:



The *App Details* page shows all of the app information, including *Services*. When you register an app, it immediately becomes available in your instance of Eloqua so you can test it before releasing it to any other prospective users. When you are ready to release your app, or just to share it with a select few, check out the app publishing tutorial.

Register an action service

With sctions, developers can build apps that control an external action that occurs when contacts enter a step on the campaign canvas. Examples of this could include sending contacts SMS messages, a direct mail piece, or registering them for webinars. When contacts enter an action step, Eloqua will call out to the app, and the app can respond by performing an action in an external system, then informing Eloqua that the process is complete. The contact will then move on in the workflow.

Actions are the next generation of Eloqua's cloud connectors framework.

When you set up an action service, you will need to provide the usual Oracle Eloqua app Developer framework service details, as well as the instance configuration and action settings.

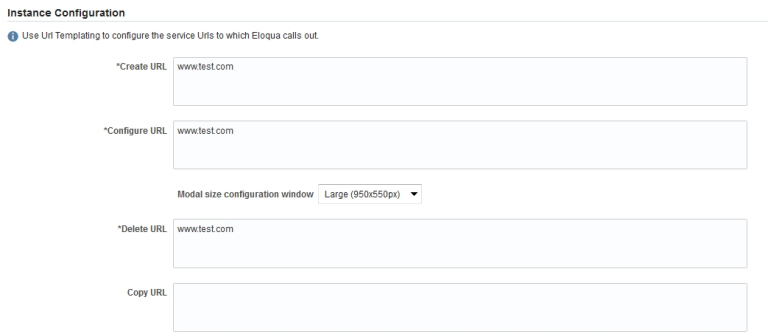
Service Details

These fields are present for all services:

- **Name**: the name of the service. Max length: 100 characters.
- **Description**: describes what the service does. Max length: 4000 characters.
- **16x16px Icon**: the URL of the icon Eloqua displays when your service is displayed on a canvas. The icon will be 16px by 16px when displayed. It must be a secure URL. See App icon design guidelines for more information on designing app icons.
- **32x32px Icon**: the URL of the icon that Eloqua should display for your service. The icon will be 32px by 32px when displayed. It must be a secure URL. See App icon design guidelines for more information on designing app icons.

Instance configuration

This section defines the URLs for creating, configuring, copying, and deleting an instance of your service. The installation tutorial details the instance creation flow when a marketer drags an Action step onto the canvas, and selects your app.
Each URL is a *templated URL* that uses common URL template parameters and some Eloqua markup language parameters. Eloqua replaces these paramters with their appropriate values when it makes a call. For more about URL templating, see our Introduction to URL templating.



- **Create URL**: A templated URL pointing to a web portal for creating an instance of this service as an HTTP `POST` request. All common URL template parameters are available.

You should be sure to include, at a minimum, the `{InstanceId}` parameter so that you will be able to identify the service in the future. On success, this endpoint should return a 200-level response with the created instance.

- **Configure URL**: A templated URL pointing to an endpoint for configuring an instance of this service as an HTTP `GET` request. All common URL template parameters are available.

  You should be sure to include, at a minimum, the `{InstanceId}` parameter so that you will be able to identify the service in the future.

- **Modal size configuration window**: Select a modal size for your configuration window. The configuration page displays when marketers configure your service.

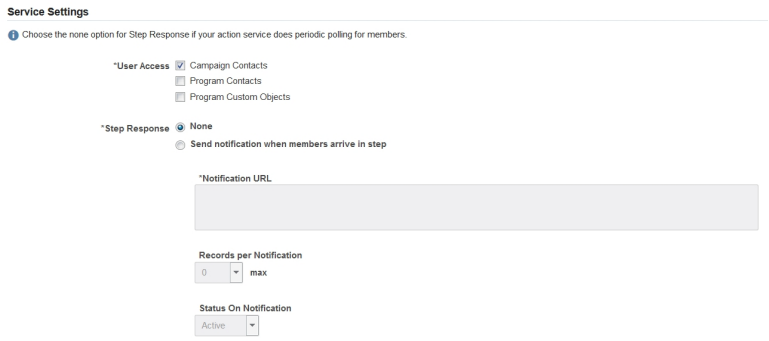
  - **Large**: 950px x 550px
  - **Small**: 650px x 550px

- **Delete URL**: A templated URL pointing to an endpoint for deleting an instance of this service using an HTTP `DELETE` request. All common URL template parameters are available. On success, this endpoint should return a 200-level response.

- **Copy URL**: A templated URL pointing to an endpoint for creating an instance of this service as an HTTP `POST` request. All common URL template parameters are available in addition to `{OriginalInstanceId}`, `{OriginalInstallId}`, `{OriginalAssetId}`, and `{OriginalAssetName}`.

  You should be sure to include, at a minimum, the `{InstanceId}` and `{OriginalInstanceId}` parameters so that you can identify the original and newly created instances. On success, this endpoint should return a 200-level response with the created instance. Learn how to Respond when a marketer copies a service instance.

Service settings



- **User access**: Specify if your service supports campaign contacts, program contacts, and/or program custom objects.
- **Step response**: Specify whether Eloqua should call out to your service when a member arrives in an Action Step, or whether you will poll for members periodically.
- **Notification URL**: This url will be called (HTTP POST) when actions are taken on an instance of this service. This endpoint should return a 2xx level response with an empty body on success. If this property is not specified, the system will fall back to a polling style approach without notification. Note that this url can be used along with the `recordDefinition` property of the instance to send data.
- **Records per notification**: Max number of records to push per HTTP request (between 0 and 5,000). If set to 0, apps must be developed to retrieve records, see the tutorial Retrieving app records using the bulk API for more information.
- **Status on Notification**: Specifies what the member's status should be set to when a notification is set. This is only valid if *max records per notification* is greater than 0. If you set the *status on notification* to **Complete**, Eloqua will call out to the notification URL when contacts flow into the action step, and will then push the contacts directly into the next step. If you set it to **Active**, you will need to call back into Eloqua to set each contact's status to complete so they will flow into the next step. See: develop an action service for more information.

When you're done configuring your service, click **Save**. A green alert message will appear at the top of the page indicating the service has been successfully saved.

Register a Decision Service

With Decisions, developers can build apps that directly control the path a contact takes as they flow through a campaign. For instance, you could build an App that Eloqua calls out to when contacts reach a Decision step. This App might interact with an external system, and evaluate rules against that set of external data. The App then responds, telling Eloqua whether the contacts should flow through the step's "yes" or "no" path. With Decisions, the data used in the decision never needs to be brought into Eloqua.

When you set up a Decision service, you will need to provide the usual AppCloud Developer framework service details, as well as the **Instance Configuration**, **Content-Type of the notification POST call** and **Decision Settings**.

Service Details

These fields are present for all services:

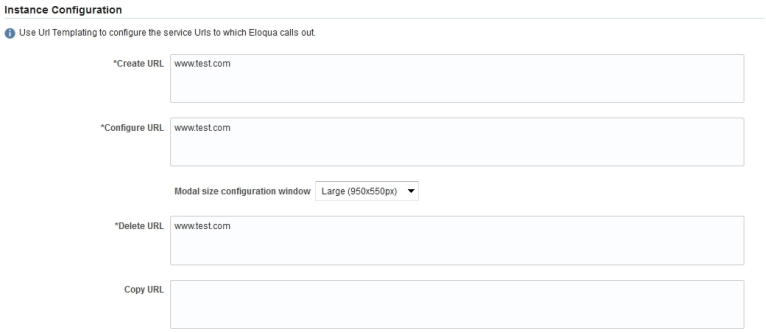
- **Name**: the name of the service. Max length: 100 characters.
- **Description**: describes what the service does. Max length: 4000 characters.
- **16x16px Icon**: the URL of the icon Eloqua displays when your service is displayed on a canvas. The icon will be 16px by 16px when displayed. It must be a secure URL. See App icon design guidelines for more information on designing app icons.
- **32x32px Icon**: the URL of the icon that Eloqua should display for your service. The icon will be 32px by 32px when displayed. It must be a secure URL. See App icon design guidelines for more information on designing app icons.

Instance Configuration

This section defines the URLs for creating, configuring, copying, and deleting an instance of your service. The installation tutorial details the instance creation flow when a marketer drags an Decision step onto the canvas, and selects your app.

Each URL is a *templated URL* that uses common URL template parameters and some Eloqua markup language parameters. Eloqua replaces these paramters with their

appropriate values when it makes a call. For more about URL templating, see our Introduction to URL templating.



- **Create URL**: A templated URL pointing to a web portal for creating an instance of this service as an HTTP `POST` request. All common URL template parameters are available.

  You should be sure to include, at a minimum, the `{InstanceId}` parameter so that you will be able to identify the service in the future. On success, this endpoint should return a 200-level response with the created instance.

- **Configure URL**: A templated URL pointing to an endpoint for configuring an instance of this service as an HTTP `GET` request. All common URL template parameters are available.

  You should be sure to include, at a minimum, the `{InstanceId}` parameter so that you will be able to identify the service in the future.

- **Modal size configuration window**: Select a modal size for your configuration window. The configuration page displays when marketers configure your service.

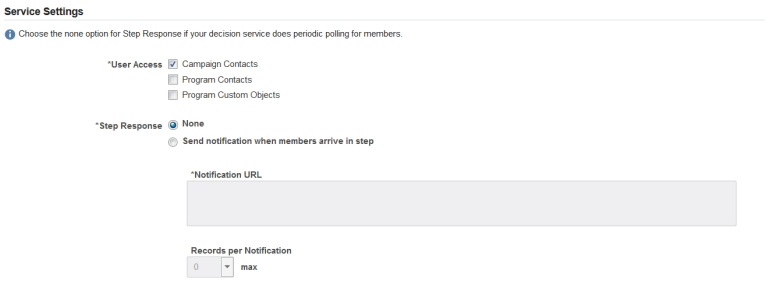
  - **Large**: 950px x 550px
  - **Small**: 650px x 550px

- **Delete URL**: A templated URL pointing to an endpoint for deleting an instance of this service using an HTTP `DELETE` request. All common URL template parameters are available. On success, this endpoint should return a 200-level response.

- **Copy URL**: A templated URL pointing to an endpoint for creating an instance of this service as an HTTP `POST` request. All common URL template parameters are available in addition to `{OriginalInstanceId}`, `{OriginalInstallId}`, `{OriginalAssetId}`, and `{OriginalAssetName}`.

  You should be sure to include, at a minimum, the `{InstanceId}` and `{OriginalInstanceId}` parameters so that you can identify the original and newly created instances. On success, this endpoint should return a 200-level response with the created instance. Learn how to Respond when a marketer copies a service instance.

Service Settings



- **User Access**: Select the Eloqua services for which your app should be available.
- **Step Response**: Specify whether Eloqua should call out to your service when a member arrives in the decision step, or whether your application will poll for members periodically.
- **Notification URL**: A templated URL to an endpoint for notifying the external service, during execution, to get the decision result of the configured instance. If this property is not specified, the system will fall back to a polling style approach without any notification.
- **Records per Notification**: Max number of records to push per HTTP request (between 0 and 5,000). If set to 0, apps must be developed to retrieve records, see the tutorial Retrieving app records using the bulk API for more information.

When you're done configuring your service, click **Save**. A green alert message will appear at the top of the page indicating the service has been successfully saved.

Register a content service

Content allows marketers to source pieces of content in Eloqua emails and landing pages, from an external source. This is the new and improved version of Eloqua's cloud components framework, and it includes many improvements such as asynchronous bulk processing (for emails), the ability to fully test the content service within Eloqua, and design-time interaction with the email and landing page editors. For example, you can now resize content in real time.

When you set up a content service, you will need to provide the usual AppCloud developer framework service details, as well as the **Instance Configuration** and **Content Settings**.

Service Details

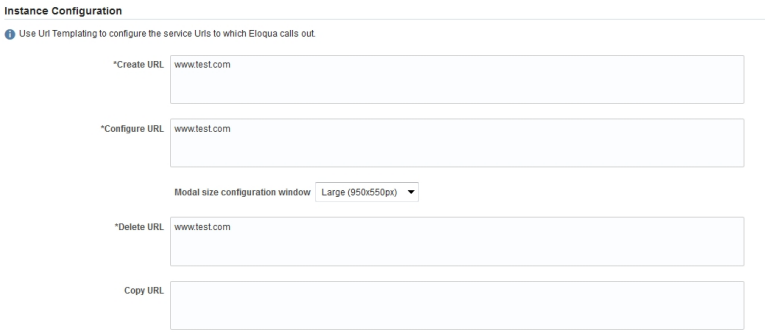
These fields are present for all services:

- **Name**: the name of the service
- **Description**: describes what the service does
- **32x32px Icon**: the URL of the icon that Eloqua should display for your service. The URL must be `https://`

Instance configuration

This section defines the URLs for creating, configuring, copying and deleting an instance of your service. The installation tutorial details the instance creation flow when a creates a landing page or email populated with content.

Each URL is a *templated URL* that uses common URL template parameters and some Eloqua markup language parameters. Eloqua replaces these paramters with their appropriate values when it makes a call. For more about URL templating, see our Introduction to URL templating.



- **Create URL**: A templated URL pointing to a web portal for creating an instance of this service as an HTTP `POST` request. All common URL template parameters are available.

  You should be sure to include, at a minimum, the `{InstanceId}` parameter so that you will be able to identify the service in the future. On success, this endpoint should return a 200-level response with the created instance.

- **Configure URL**: A templated URL pointing to an endpoint for configuring an instance of this service as an HTTP `GET` request. All common URL template parameters are available.

  You should be sure to include, at a minimum, the `{InstanceId}` parameter so that you will be able to identify the service in the future.

- **Modal size configuration window**: Select a modal size for your configuration window. The configuration page displays when marketers configure your service.

  - **Large**: 950px x 550px
  - **Small**: 650px x 550px

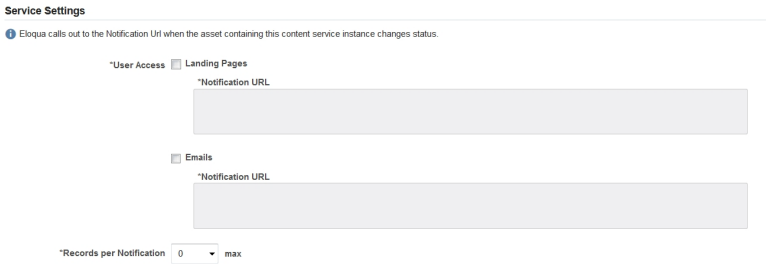
- **Delete URL**: A templated URL pointing to an endpoint for deleting an instance of this service using an HTTP `DELETE` request. All common URL template parameters are available. On success, this endpoint should return a 200-level response.

- **Copy URL**: A templated URL pointing to an endpoint for creating an instance of this service as an HTTP `POST` request. All common URL template parameters are available in addition to `{OriginalInstanceId}`, `{OriginalInstallId}`, `{OriginalAssetId}`,

and {OriginalAssetName}.

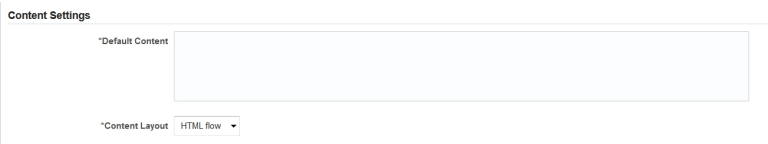
You should be sure to include, at a minimum, the {InstanceId} and {OriginalInstanceId} parameters so that you can identify the original and newly created instances. On success, this endpoint should return a 200-level response with the created instance. Learn how to Respond when a marketer copies a service instance.

Service settings



- **User Access**: Select the Eloqua asset types for which your app should be available.
  - **Landing Page Notification (HTML) URL**: templated URL that Landing Page engines should call out to during render to retrieve the HTML content for the configured instance. In addition to the common HTML template parameters, the notification (HTML) URL supports the {VisitorId}, {Fragment}, {Host}, {Query} and {Url} parameters. Be sure to include, at a minimum, the the {InstanceId} and {ExecutionId} parameters so that you will be able to call back in to Eloqua.
  - **Email Notification (HTML) URL**: templated URL that the email engine should call out to during render to retrieve the HTML content for the configured instance. Be sure to include, at a minimum, the the {InstanceId} and {ExecutionId} parameters so that you will be able to call back in to Eloqua.
- **Max Records per Notification**: max number of records to push per HTTP request (between 0 and 5,000)

Content settings



- **Default Content**: the HTML content the system should use if the external service is not accessible. This is a required field and cannot be empty.
- **Content Layout**: choose from flow or fixed. If you choose flow, the external content is input into your landing page or email with no constraints (except width). If you choose fixed, the external content will be constrained in height and width, as specified by the content instance, and will not be configurable from within Eloqua.

If the landing page or email notification URL is not specified and a landing page or email rendering is required, the default content is used instead.

When you're done configuring your service, click **Save**. A green alert message will appear at the top of the page indicating the service has been successfully saved.

Register a feeder service

With feeders, developers can build apps that use data in third-party services to drive campaign membership. When you set up a feeder service, you will need to provide the usual AppCloud developer framework service details, as well as the **Instance Configuration** and **Feeder Settings**.

Service Details

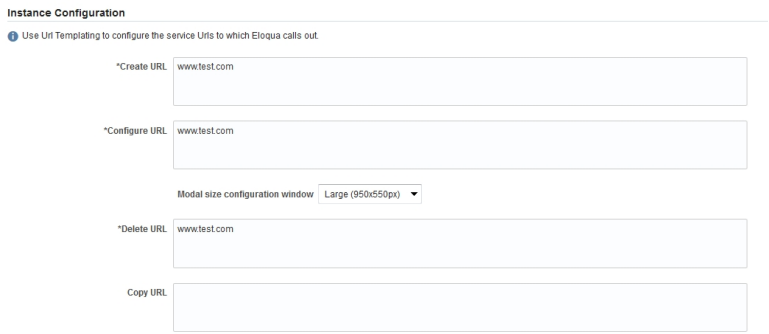
These fields are present for all services:

- **Name**: the name of the service. Max length: 100 characters.
- **Description**: describes what the service does. Max length: 4000 characters.

- **16x16px Icon**: the URL of the icon Eloqua displays when your service is displayed on a canvas. The icon will be 16px by 16px when displayed. It must be a secure URL. See App icon design guidelines for more information on designing app icons.
- **32x32px Icon**: the URL of the icon that Eloqua should display for your service. The icon will be 32px by 32px when displayed. It must be a secure URL. See App icon design guidelines for more information on designing app icons.

Instance configuration

This section defines the URLs for creating, configuring, copying, and deleting an instance of your service. The installation tutorial details the instance creation flow when a marketer drags an feeder step onto the canvas, and selects your app.
Each URL is a *templated URL* that uses common URL template parameters and some Eloqua markup language parameters. Eloqua replaces these paramters with their appropriate values when it makes a call. For more about URL templating, see our Introduction to URL templating.



- **Create URL**: A templated URL pointing to a web portal for creating an instance of this service as an HTTP `POST` request. All common URL template parameters are available.

  You should be sure to include, at a minimum, the `{InstanceId}` parameter so that you will be able to identify the service in the future. On success, this endpoint should return a 200-level response with the created instance.

- **Configure URL**: A templated URL pointing to an endpoint for configuring an instance of this service as an HTTP `GET` request. All common URL template

parameters are available.

You should be sure to include, at a minimum, the `{InstanceId}` parameter so that you will be able to identify the service in the future.

- **Modal size configuration window**: Select a modal size for your configuration window. The configuration page displays when marketers configure your service.
    - **Large**: 950px x 550px
    - **Small**: 650px x 550px
- **Delete URL**: A templated URL pointing to an endpoint for deleting an instance of this service using an HTTP `DELETE` request. All common URL template parameters are available. On success, this endpoint should return a 200-level response.
- **Copy URL**: A templated URL pointing to an endpoint for creating an instance of this service as an HTTP `POST` request. All common URL template parameters are available in addition to `{OriginalInstanceId}`, `{OriginalInstallId}`, `{OriginalAssetId}`, and `{OriginalAssetName}`.

  You should be sure to include, at a minimum, the `{InstanceId}` and `{OriginalInstanceId}` parameters so that you can identify the original and newly created instances. On success, this endpoint should return a 200-level response with the created instance. Learn how to Respond when a marketer copies a service instance.

Feeder settings

- **User Access**: Select the Eloqua services for which your app will be available.
- **Notification URL**: This URL is called (HTTP `POST`) when a campaign containing an instance of the service's status changes, such as when it is activated.

Be sure to include the `{AssetId}` and `{EventType}` parameters in the templated URL. `AssetId` describes the campaign whose status has changed, and `EventType` defines the status. When a campaign is activated, for example `EventType` will be “Active”. Otherwise, you will not be able to identify what the incoming call references.

When you’re done configuring your service, click **Save**. A green alert message will appear at the top of the page indicating the service has been successfully saved.

Register a menu service

Eloqua's menus service is all about *context*. With menus, you can build an App that enables a marketer to launch an externally-hosted application from within Eloqua via a menu dock. This menu dock floats on the right side of the screen in the Eloqua's campaign canvas or the asset editor screens. When a marketer clicks on your menu app, Eloqua calls out to your service on the specified endpoint URL, passing along information about the asset or area from which your App was launched.

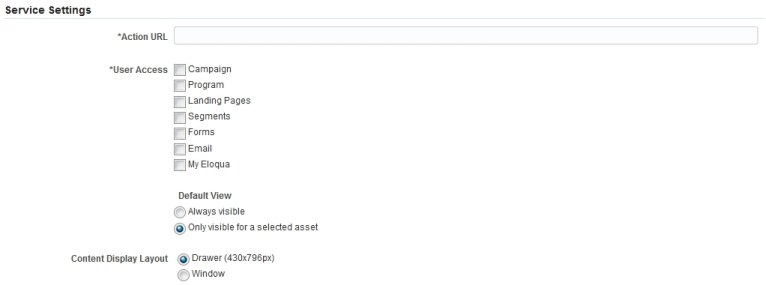
When setting up a menu service, you will need to provide basic service details, as well as the **Menu Extension Service Details**.

Service Details

These fields are present for all services:

- **Name**: the name of the service
- **Description**: describes what the service does
- **32x32px Icon**: the URL of the icon that Eloqua should display for your service. The URL must be `https://`

Service settings



- **Action URL**: A templated URL that defines where users are redirected to when they select the menu item. For example, a new tab with an external page showing information on a Campaign.
- **User Access**: Select the assets/areas within Eloqua that your app will appear to the Marketer.
- **Default View**: Specify how your service will appear.
    - **Always visible**: Display the menu service in two areas: within the selected area of the interface and within the editor for the asset.

      > **Example**: If this option is selected along with *Campaign* access, a user will see the menu service when they navigate to **Orchestration** > **Campaigns**, and when they open a campaign.

    - **Only visible for a selected asset**: Display the menu service only for the selected assets under user access.

      > **Example**: If this option is selected along with *Campaign* access, a user will only see the menu service when they open a campaign.

- **Content Display Layout**: Select a layout option for your content display.

- **Drawer**: Content will open within the AppCloud menu dock.
- **Window**: Content will open in a new window or tab.

When you are done configuring your service, click **Save**. A green alert message will appear at the top of the page indicating the service has been successfully saved.

Register a Firehose Service

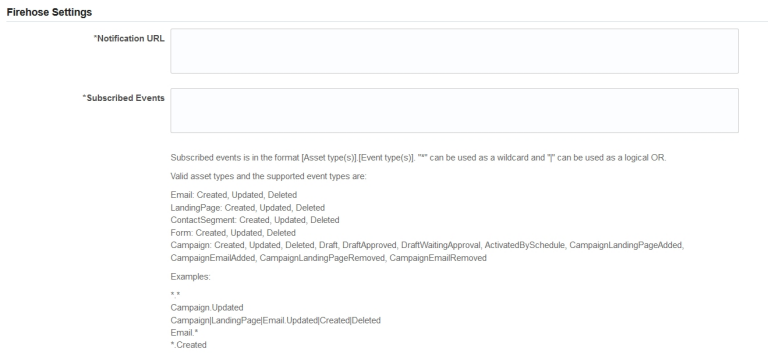
Firehose enables third-parties to build a service that receives a notification when a marketer performs an action in Eloqua. This is essentially a web-hook that will be triggered when a marketer does things like creating or editing a campaign, or creating or updating an email etc.

Service Details

These fields are present for all services:

- **Name**: the name of the service
- **Description**: describes what the service does
- **32x32px Icon**: the URL of the icon that Eloqua should display for your service. The URL must be `https://`

Firehose Settings



- **Notification URL**: the URL that Eloqua will call out to with the notification. This is a templated URL.
- **Subscribed Events**: defines which events trigger a call out to the external application. The service configuration page includes a list of valid asset types and the event types supported for each asset, as well as examples of valid patterns.

When you're done configuring your service, click **Save**. A green alert message will appear at the top of the page indicating the service has been successfully saved.

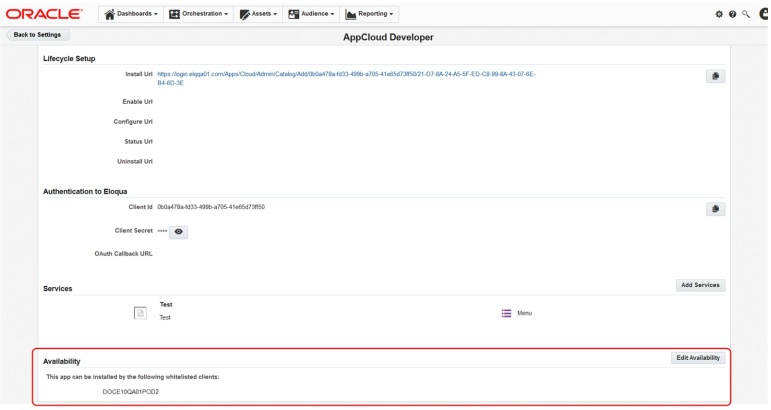
Publish your app

The final step in the app configuration lifecycle is publishing your app. The publishing flow has two components: whitelisting users so that they are permitted to install the app, and retrieving the link that those users need in order to add the app to their catalog.

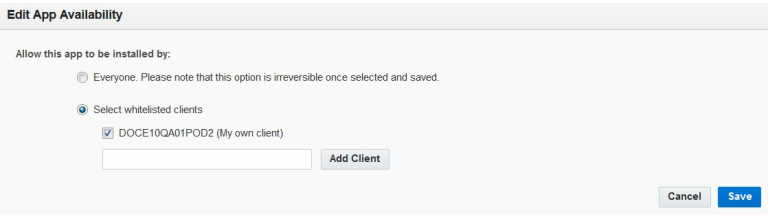
Grant access

By default, when you register an app with Eloqua, only your instance of Eloqua has access to it. This makes it easy for you to test your app to ensure it is behaving as expected.

To allow other people access to the app, you have to grant access from the app Configuration page, by clicking **Edit Availability** in the **Availability** section.



You can choose between granting access to all Eloqua instances, or select Eloqua instances.



Publish to all sites

If your app is available to all Eloqua instances, all sites are immediately whitelisted. At present, this cannot be undone. Nevertheless, a customer admin will need the install URL before they can actually install the app.

Publish to a specific site

To make your app available to specific Eloqua instances, enter the name of the site you want to add and click **Add Client**. All Eloqua Instances with that site name will now be able to install the app.

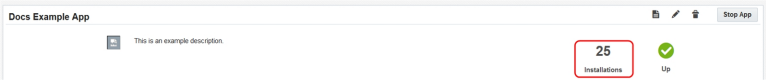
> **Note**: You cannot remove a site from your app's whitelist if that site has your app installed. To remove a site from your app's whitelist, you must first contact the appropriate client and have them uninstall your app. This a security measure in place to prevent broken assets and campaigns.

Share the URL

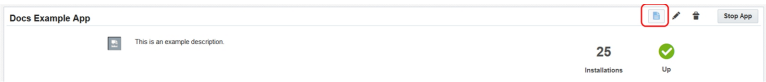
Having published your app, giving sites the ability to install it, you now need to share the URL so that users can add the app to their Eloqua instance's app Catalog, giving marketers the ability to use it. Eloqua automatically creates the installation URL when you register your app.

Many developers send the link by email to the sites they have authorized. When an authorized user clicks the link, the app is added to their catalog. The link will not work for users who are not authorized, so you do not need to worry about people passing around the installation link to sites you did not whitelist.

The number of sites which have installed your app is listed in the app details page.



Click **Logs** to see catalog install details.



Next steps

When a customer admin installs an app on a new site, Eloqua calls out to the app's install URL. Refer to the AppCloud Installation Flow tutorial for more information. You may also want to publish your app to the Oracle Cloud Marketplace.

AppCloud installation flow

When a Customer Admin installs an app, Eloqua calls out to the configured **enable URL** with a POST request. The call will resemble the following:

```
https://example.com/awesomeapp/api/enable?enableurl&oauth_consumerkey=XXX-XXX-XXX-XXX-XXX
&oauth_nonce=nonce&oauth_signature_method=SIG-METH&oauth_timestamp=111111111
&oauth_version=1.0&oauth_
signature=signature&callback=https%3A%2F%2Fsecure.eloqua.com%2FApps%2FCloud%2FAdmin%2FInstall%2FCallback%2Fa1b2c3d4%3Fguid%3Dz9y8x7w6
```

The app must respond in one of two ways: if the app does not require configuring by the user, it can return an HTTP 200 OK response. Otherwise, it should respond with an HTTP 302 redirect to another page. The following sections detail each option.

No additional information needed

In this scenario, there is no need to obtain OAuth authorization for the installing user's instance, nor any other configuration information. The appropriate response is then:

```
HTTP/1.1 200 OK
```

The app install status will be set to the "Ready" and marketers will be able to use it.

Additional information needed

If you need to have the user configure the application or provide OAuth authorization for their instance, you should respond with an HTTP 302 redirect to another page. When the app is configured, you need to call back in to the callback URL.

1. Respond with a redirect:

   ```
   HTTP/1.1 302 Redirect
   ```

   If your app needed to go through the OAuth flow, the redirect might point to `login.eloqua.com/auth/oauth2/authorize`, Eloqua's OAuth endpoint. If you needed users to configure options in the app when they install it, the redirect might point to a configure page within your app.

2. Call in to the **callback URL** when the user has finished configuring the application. Using the above example, the call would look like:

   ```
   https://secure.eloqua.com/Apps/Cloud/Admin/Install/Callback/a1b2c3d4?guid=z9y8x7w6
   ```

   Eloqua provides the **callback URL** when you request it as a template in your templated **enable URL**. Calling the **callback URL** tells Eloqua that the install status should be set to "Ready" so that marketers can start using the application.

3. When the app provider calls back in to complete the installation, it can optionally append a redirect URL:

   ```
   https://secure.eloqua.com/Apps/Cloud/Admin/Install/Callback/{installId}?guid={guid}&redirect={redirectUrl}
   ```

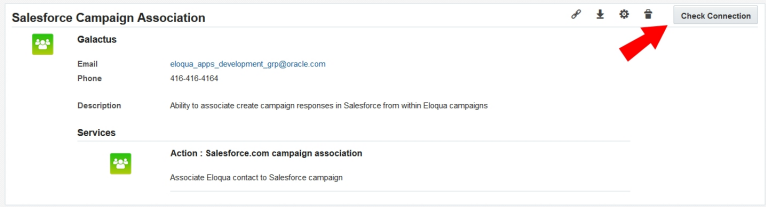
   Replace parameters in {} with desired values, and the redirect URL must be in HTTP or HTTPS. The redirect URL supports templated URL parameters. You must

URL encode the redirect URL to ensure that template parameters are returned back to you. If no redirect URL is given, or if the redirect URL given is not acceptable, then the user is redirected to Eloqua's catalog.

Respond when Eloqua calls the status URL

When you register an app in Eloqua, you are able to specify a status URL. This URL is an endpoint which Eloqua can call to verify that your app is up and running.

Eloqua calls the status URL when a user clicks the **Check Connection** button for that app from the AppCloud catalog.



Important: Applications have an application status associated with them. This status is either up, down, or maintenance. Calls to the status URL do not update the application status: they merely display the application's response to the user who clicked the **Status** button.

Example

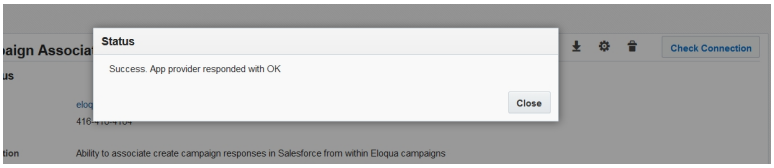
Suppose the user clicked the **Check Connection** button for AwesomeApp. The call resembles:

```
GET https://example.com/awesomeapp/status
```

If AwesomeApp is operating normally, the appropriate response is an 200-level response:

```
HTTP/1.1 200 OK
```

Eloqua then displays the status to the user in a modal window.



Otherwise, if AwesomeApp is down and responds with a 400 or 500-level message, Eloqua indicates that it was unable to connect to the application. Or that the application is in maintenance mode.

Respond when a marketer copies a service instance

Marketers often create new campaigns, landing pages, and emails from existing resources. When a marketer creates a campaign, landing page, or email using the **Save As** option in the campaign canvas and landing page or email editors all of the objects in that resource are duplicated. For most items on a canvas, this is straightforward. For services that follow the instantiation-execution model, however, each instance has its own unique identifier. Thus, when an action, decision, feeder, or content service instance is copied, a new GUID needs to be assigned to the copy. The Oracle Eloqua app developer framework uses the **Copy URL** to communicate with the service's provider to configure and update the new service instance when a service is copied.

Sample copy URL call and response

Eloqua's call to the copy URL is similar to the call to the create URL when a service is first created. The following section provides step-by-step instructions for properly handling calls to the copy URL. This example uses a decision service, but the process is the same for actions, content, and feeders.

1. A marketing user copies a campaign that includes an AwesomeApp decision service whose templated copy URL is:

```
example.com/awesomeapp/decide/copy?instanceId={InstanceId}&originalId=
{OriginalInstanceId}
```

2. Eloqua calls out to the copy URL, replacing the templated parameters with the original service's GUID, and the new GUID of the copy. AwesomeApp can use the original service's GUID to determine the appropriate configuration options for the copy:

```
POST example.com/awesomeapp/decide/copy?instanceId=bcbdff6a-74dd-41c9-b716-
825e4049b776
&originalId=d155213e-cd30-422b-984c-acb33093cb5b
```

3. AwesomeApp responds in one of two ways: either it replies with an HTTP 200 Ok response, and no content, in which case Eloqua will use the original service's configuration for the new service, or, AwesomeApp replies with a 200-level HTTP response along with a DTO specifying how the new service should be configured, as in the following:

```
HTTP/1.1 200 OK
{
  "recordDefinition":
    {
      "ContactId": "{{Contact.Id}}",
      "EmailAddress": "{{Contact.Field(C_EmailAddress)}}"
    }
}
```

Internally, Eloqua then updates the references in the campaign to the new service instance GUID, and the new campaign is created.

Persistent settings and data

As an app provider, you have control over what data and settings are persisted when a service is copied: if you wish to retain the configuration of the original service, simply return a blank DTO when Eloqua calls out to the copy URL. Conversely, if you wish to update the configuration of the new service, you can do so by replying with a new DTO.

Once a service has been copied, and the new service has been configured, there is ***no link*** between the original and new services: changing the configuration in one does not affect the other.

Scheduled maintenance

If you need to perform maintenance on your app, and wish to put it into maintenance mode, you can do so directly from the app's *App Details* page from within your Eloqua instance by clicking the **Stop App** button.

When an app is in maintenance mode:

- Eloqua will not send any calls to the app.
- Marketers should not be able to add the app's services to a campaign, email, or landing page.
- When a user checks the app's status, Eloqua will indicate the app is undergoing maintenance.

App shutdown is not equivalent to deactivating or deleting an app. It is useful if you wish to suspend the app while performing an upgrade.

App shutdown

If your app is not operating properly, Oracle Eloqua can shut down your app. This prevents marketers from creating new campaigns that use your app's services. Marketers can also configure a default path for contacts to follow in the event that a service is unavailable while a campaign is in progress. See: configuring cloud action and decision

elements in the Eloqua user help for more information. When an app is shutdown, any contacts in the app's service's steps will flow down the default path, if it is configured.

Shutdown

Eloqua will shut down your app if there were **more than 100 calls in the last five minutes, and 25% of them failed**. Eloqua will automatically send an email message to the email address associated with your provider, with a link you can use to reactivate your app. When the app is shut down, all pending and future requests are marked as "failed".

Reactivate your app

To reactivate your app, click on the link in the notification email that Eloqua sent. Alternatively, log in to your development instance and click **Start App** on the *App Details* page of Eloqua's *AppCloud Developer Framework* area.

You can programatically start or stop your app using the `GET /apps/{appid}/configurations` and `PUT /apps/{appid}/configurations` endpoints. See: programmatically update your app's status for more information.

App icon design guidelines

When designing your app's icon, it is important to follow the recommended design guidelines to ensure your app icon displays correctly for your users. Each app requires:

- AppCloud catalog icon
- App-specific icon (the size requirement depends on where the app appears in the Eloqua interface)
    - Menu
    - Content
    - Firehose
    - Canvas

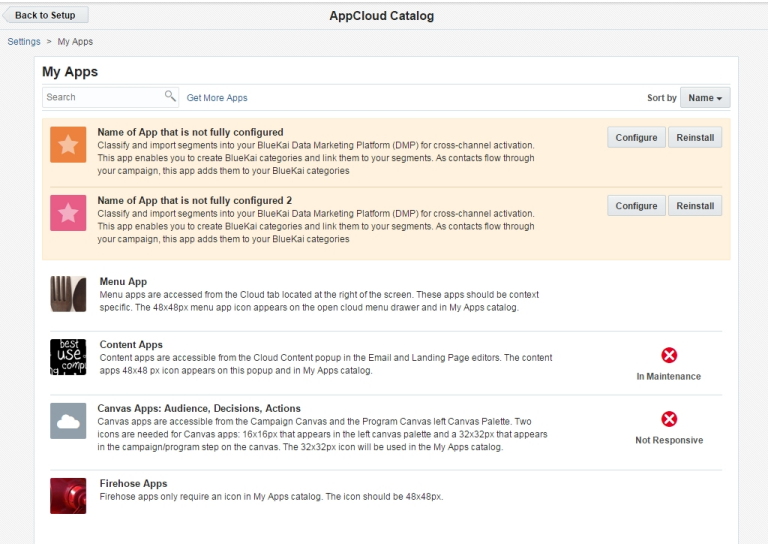
Each app has different size guidelines and will appear in different areas of the Eloqua interface:

AppCloud catalog icons

Size requirement:

- 96x96px

An icon to represent your app will be displayed in the AppCloud catalog and on the *App Details* page.

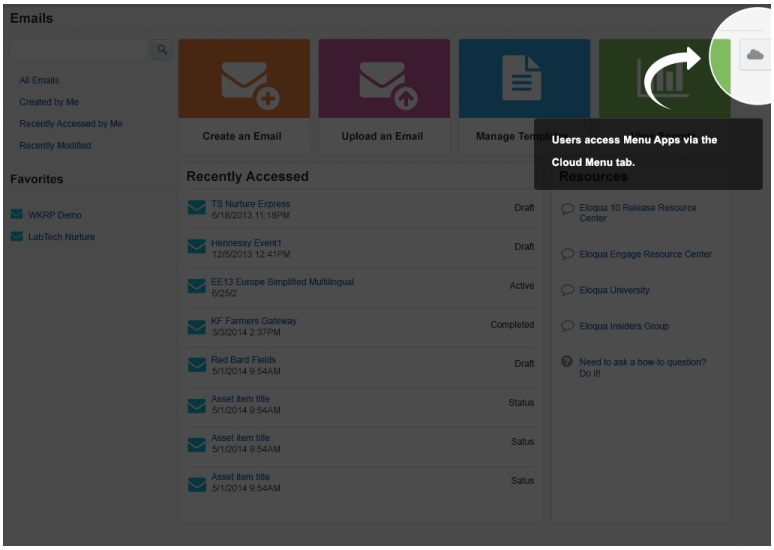


Menu apps

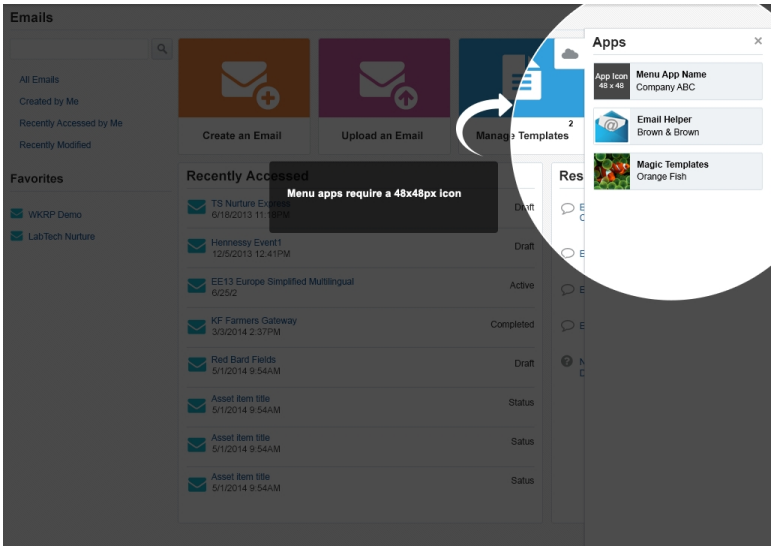
Size requirement:

- 96x96px

Menu apps are accessed from the cloud menu tab, located at the top right of the screen. These apps should be context specific.



The menu app icon appears within the cloud menu tab and in AppCloud catalog.

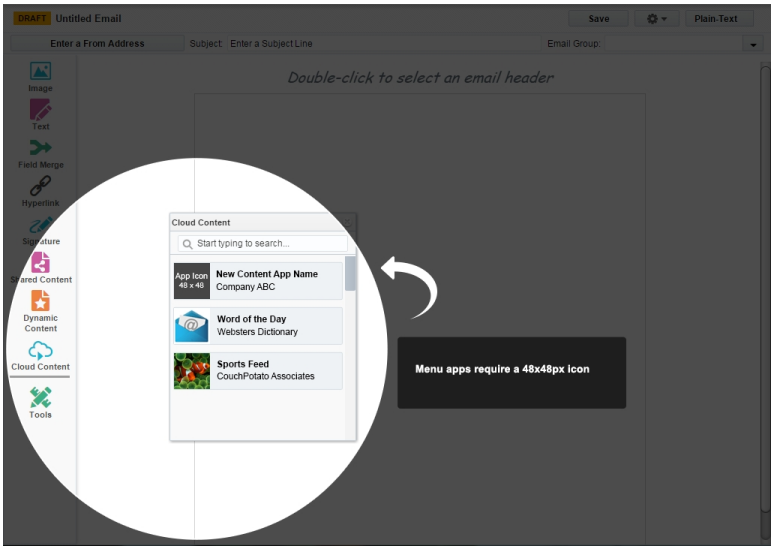


Content apps

Size requirement:

- 96x96px

Content apps are accessible from the cloud content popup in the email and landing page editors. The content apps icon appears on this popup and in the AppCloud catalog.

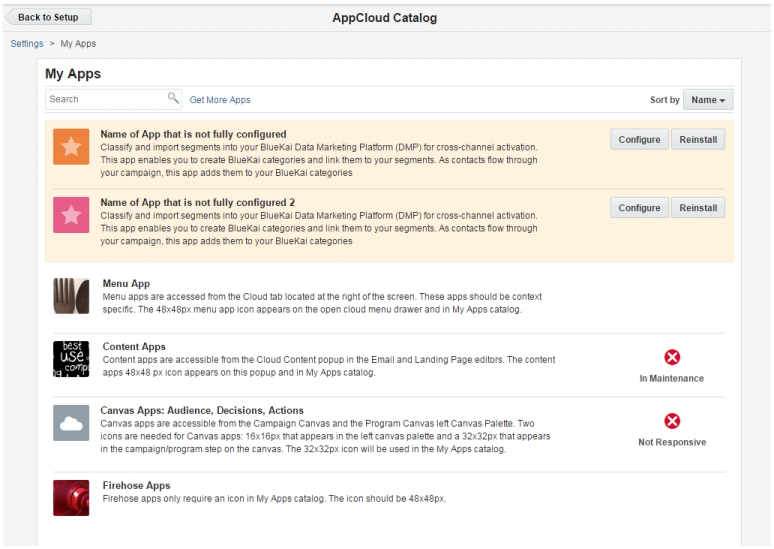


Firehose apps

Size requirement:

- 96x96px

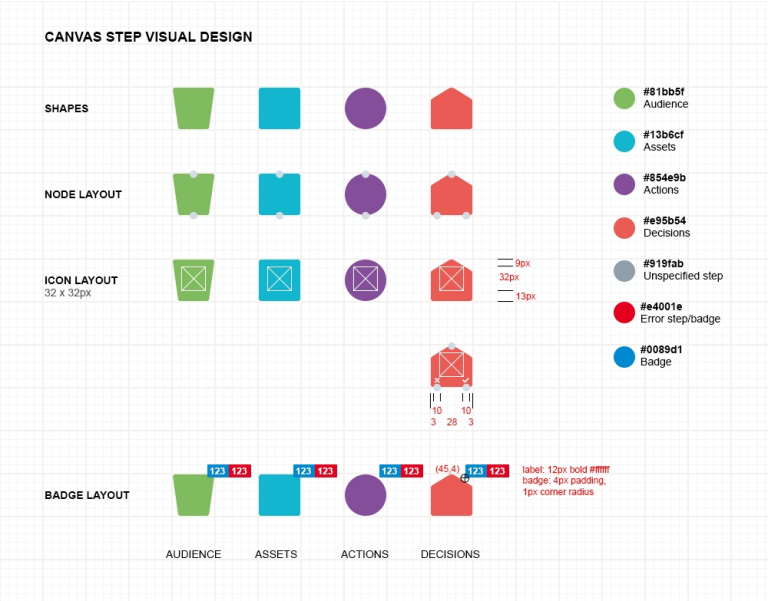
Firehose apps only require an icon in the AppCloud catalog.



Canvas apps (action, decision, and feeder services)

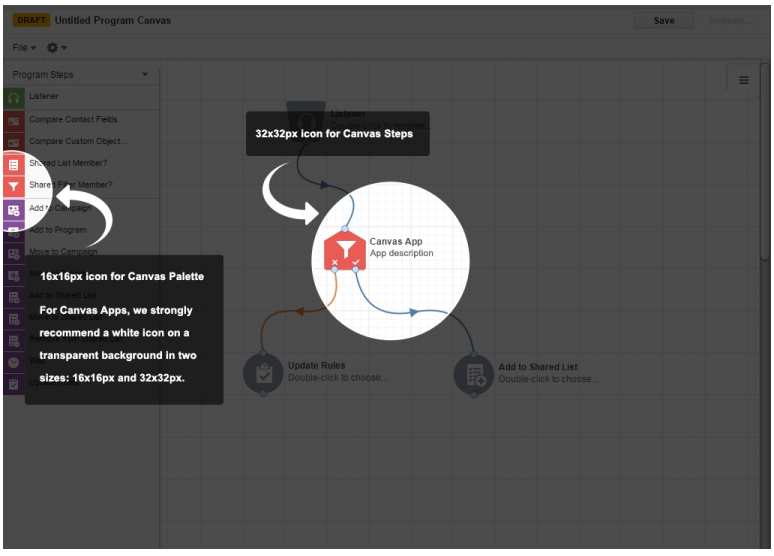
Size requirement:

- 32x32px
- 16x16px



Canvas apps are accessible from the campaign canvas and the program canvas left canvas palette. Two icons are needed for canvas apps:

- The 32x32px icon appears in the left campaign/program step on the canvas
- The 16x16px icon appears in the left canvas palette



Recommendation:

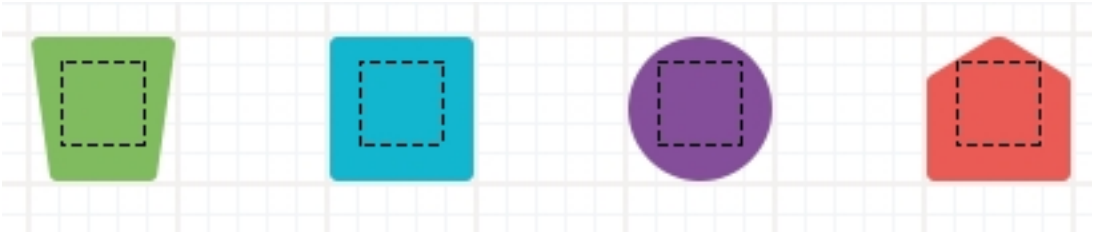
- White icon with transparent background

> **Important**: Because color is used to differentiate between audience (green), decision (red) and action (purple) steps and unpopulated state steps (gray), it is strongly recommended that app providers use white icons with transparent backgrounds for canvas apps.

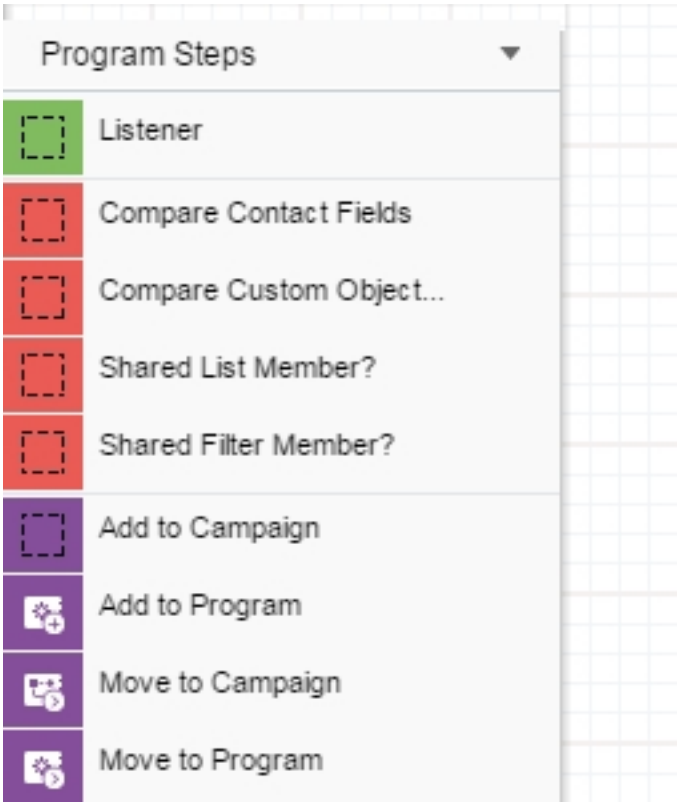
Migrating existing apps to the new interface

In the new interface, your existing apps will be modified accordingly:

- **Menu apps**: Unchanged, but scale in size.
- **Content apps**: Unchanged, but scale in size.
- **Firehose apps**: Unchanged, but scale in size.
- **Canvas apps**: The dotted line shows placement of the icon within the step:



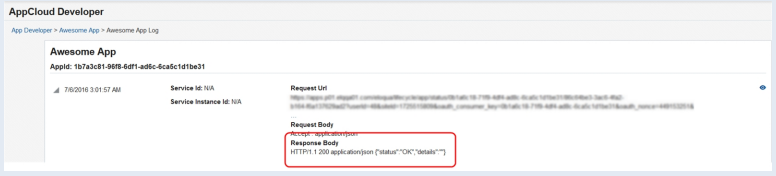
Left palette:



Viewing an app's outbound logs

The logs page records all the outbound requests made from Eloqua to your app's lifecycle URLs (the URLs an app calls when the app is configured, uninstalled, and so on). The records in the outbound log are useful for determining if Eloqua is calling your app's lifecycle URLs successfully.

Example: You can review the logs to determine if Eloqua is successfully calling the *Enable URL* by searching the logs page for the URL and viewing the response body to ensure a 200 level response code is returned.

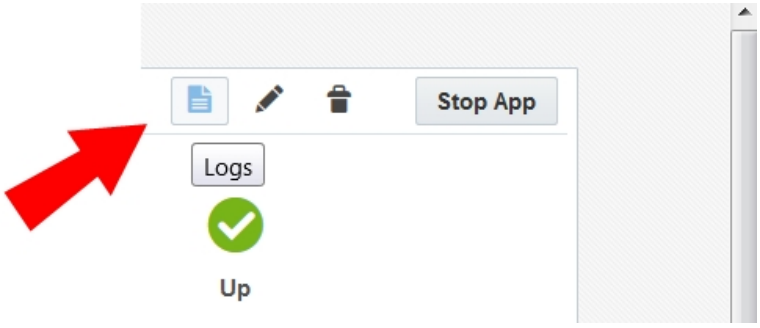


If Eloqua fails to call a URL, a 400 level response code is returned.

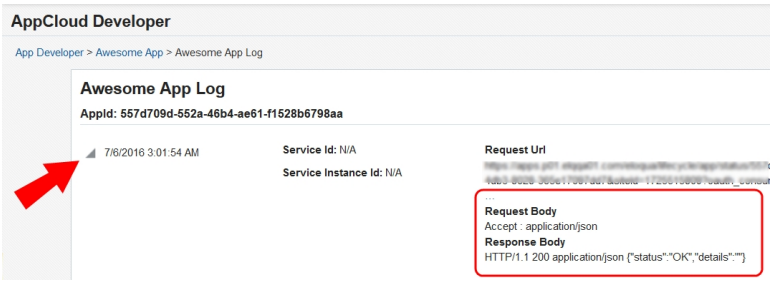
The logs page only displays outbound requests for your instance and other clients on your pod, to pull logs from another pod, use the app logs endpoint.

To view an app's outbound logs:

1. Navigate to **Settings** > **AppCloud Developer**.
2. Select your app and click **Logs**.

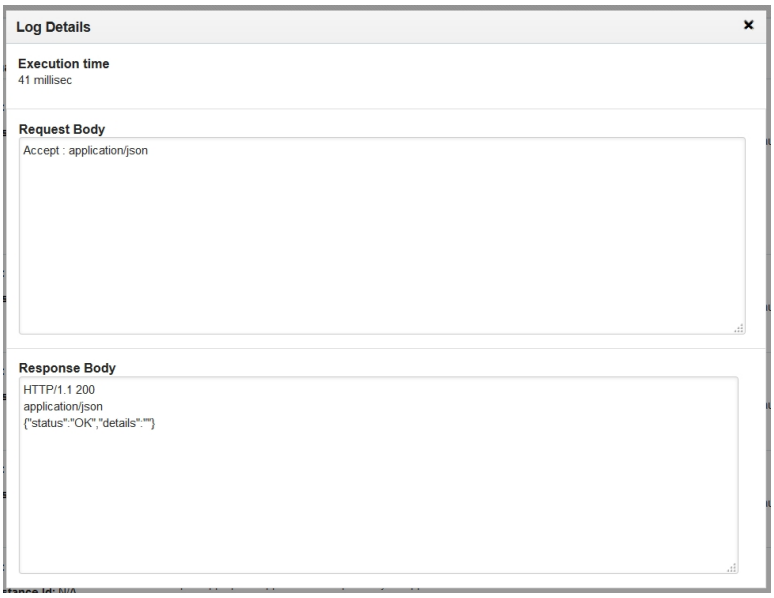


Expand the chevron to see the request and response body for the requests made to the lifecycle URLs.



Click **Log Details** to view details about the request.

The **Log Details** display how long the request took to execute, as well as the full request and response body.



Update or check your app's status using the cloud API

From time to time, you may wish to update your App's status. While it is possible to start or stop your app from the *App Details* page in the *AppCloud Developer Framework* area of your Eloqua development instance, you can also do so programmatically using a cloud API endpoint. The endpoint also enables you to check the status of your app. You will need to know your app's ID in order to call out to the cloud API endpoints. The app ID is listed on the *App Details* page in your Eloqua development instance, and is a GUID.

The cloud API URL is `https://<eloqua base url>.com/api/cloud/1.0/<endpoint>`. All calls to the API must be authenticated using OAuth. Learn more about configuring and authenticating with OAuth.

Checking the app's status

Retrieve your app's status using the `GET /apps/{id}/configurations` endpoint:

```
GET /apps/a0f6779e-8263-4bfb-a292-9714cd10d1e7/configurations
```

The response should resemble:

```
HTTP/1.1 200 OK
{
   "uri": "/apps/a0f6779e-8263-4bfb-a292-9714cd10d1e7/configurations",
   "clientSecret":
"a1ba63f9aae1cc84d91544f15b0af27ab6ced4e42c6145b9cc0425742ecf2a0da1ba63f9aa
e1cc84d91544f15b0af27ab6ce",
   "name": "Call Example",
   "description": "This is a test",
   "shortDescription": "This is a test",
   "iconUrl": "https://www.example.com/logo.png",
   "enableUrl": "http://example.com/enable/{appId}/{installId}/{userName}/{siteName}/
{siteId}",
   "statusUrl": "http://example.com/status/{installId}",
   "callbackUrl": "http://example.com/callback/{installId}",
   "makeAvailableUrl": "https://example.com",
   "publishedSites": [
      "AwesomeCompany"
   ],
   "status": "Up"
}
```

The `status` field indicates the app's current status, either "Up", "Down", or "Maintenance".

To update the app's status

Update the status by sending a request to the `PUT /apps/{id}/configurations` endpoint.

> **Important**: You must include all of the fields in the configuration document in the `PUT` request. You cannot merely specify a value for the `status` field.

The `status` can be "Up", "Down", or "Maintenance":

```
PUT /apps/a0f6779e-8263-4bfb-a292-9714cd10d1e7/configurations
{
   "uri": "/apps/a0f6779e-8263-4bfb-a292-9714cd10d1e7/configurations",
   "clientSecret":
```

```
"a1ba63f9aae1cc84d91544f15b0af27ab6ced4e42c6145b9cc0425742ecf2a0da1ba63f9aa
e1cc84d91544f15b0af27ab6ce",
   "name": "Call Example",
   "description": "This is a test",
   "shortDescription": "This is a test",
   "iconUrl": "https://www.example.com/logo.png",
   "enableUrl": "http://example.com/enable/{appId}/{installId}/{userName}/{siteName}/
{siteId}",
   "statusUrl": "http://example.com/status/{installId}",
   "callbackUrl": "http://example.com/callback/{installId}",
   "makeAvailableUrl": "https://example.com",
   "publishedSites": [
      "AwesomeCompany"
   ],
   "status": "Maintenance"
}
```

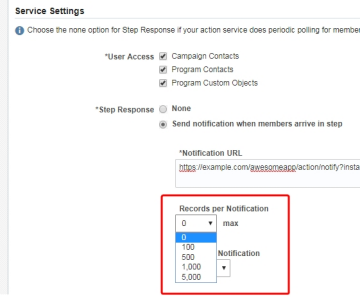
If successful, Eloqua will respond with an `HTTP/1.1 204 No Content`.

If you `GET` the `/apps/{id}/configurations` endpoint again, the response should reflect the updated status.

Retrieving app records using the bulk API

You can get notified when contacts or custom objects arrive to your app within a campaign or program. Eloqua will send your app a notification call in the form of a HTTP POST request that includes data about the contacts or custom object records. This is accomplished by supplying a **Notification URL** during service registration for action and decision services.

You control the maximum amount of contacts or custom object records sent per notification call by setting the **Records per Notification** option during service registration.



When **Records per Notification** is greater than 0, the HTTP POST request sent to the app will include data for the contacts or custom objects within the `items` array.

See an example

```
POST https://example.com/awesomeapp/action/notify?instance=
{InstanceId}&asset={AssetId}&execution={ExecutionId}

{
   "offset" : 0,
   "limit" : 1000,
   "totalResults" : 2,
   "count" : 2,
   "hasMore" : false,
   "items" :
   [
     {
       "ContactID" : "1",
       "EmailAddress" : "fred@example.com",
       "field1" : "stuff",
       "field2" : "things",
       "field3" : "et cetera"
     },
     {
       "ContactID" : "2",
       "EmailAddress" : "john@example.com",
       "field1" : "more stuff",
       "field2" : "other things",
       "field3" : "and so on"
     }
   ]
}
```

This request can get large, especially if the maximum records sent per notification is 5,000 for example. It is possible instead for app developers to select the maximum amount of **Records per Notification** to 0 and retrieve records on their own.

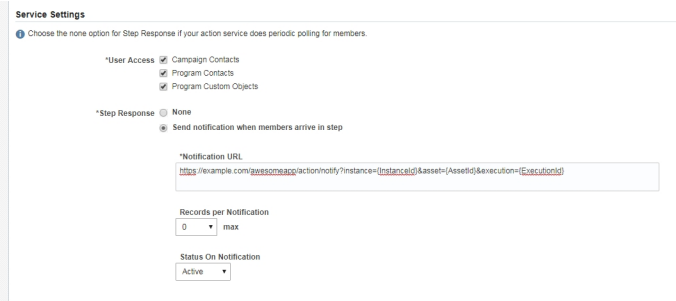
Setting records per notification to 0

If **Records per Notification** is set to 0, the notification call is sent with no data within the `items` property. The developer must then retrieve the records within the step using a separate bulk API export. This enables developers to control when and how quickly records are retrieved.

This tutorial will walkthrough the flow of having **Records per Notification** set to 0 and guiding developers through the flow of retrieving records within a step using a separate bulk API export.

Eloqua app service settings

Before we walkthrough how to respond to the notification call, note the Eloqua app service settings we will be using for the examples in this tutorial.



- Note the notification URL contains the 3 template parameters: `InstanceId` , `AssetId`, and `ExecutionId`.
- **Records per Notification** is set to **0**.

To edit your service settings, navigate to **Settings** > **AppCloud Developer** and edit your service's settings.

Notification call

When contacts or custom objects arrive in the step, Eloqua sends a HTTP POST request to the app's Notification URL.

```
POST https://example.com/awesomeapp/action/notify?instance=f82d50cd-86a9-4fca-b37e-4ec9a98b0339&asset=456&execution=12345

{
   "offset" : 0,
   "limit" : 1000,
   "totalResults" : 0,
   "count" : 0,
   "hasMore" : false,
   "items" : []
```

```
}
```

Developers must then create a bulk API export definition using the data from Eloqua's call to the Notification URL.

Using contacts as an example, let's walkthrough how to form our bulk API export definition to retrieve the records.

Request

```
POST https://secure.p03.eloqua.com/api/bulk/2.0/contacts/exports

{
   "name": "Awesome App Contact Export - f82d50cd-86a9-4fca-b37e-
4ec9a98b0339 - 12345",
   "fields": {
      "emailAddress": "{{Contact.Field(C_EmailAddress)}}"
   },
     "filter": "STATUS('{{ActionInstance
(f82d50cd86a94fcab37e4ec9a98b0339).Execution[12345]}}') = 'pending'"
}
```

Response

```
{
   "name": "Awesome App Contact Export - f82d50cd-86a9-4fca-b37e-
4ec9a98b0339 - 12345",
   "fields": {
      "emailAddress": "{{Contact.Field(C_EmailAddress)}}"
   },
   "dataRetentionDuration": "PT12H",
   "uri": "/contacts/exports/55",
   "createdBy": "API.User",
   "createdAt": "2018-08-19T20:51:28.8201911Z",
   "updatedBy": "API.User",
   "updatedAt": "2018-08-19T20:51:28.8201911Z"
}
```

Next, create a sync using the export definition uri.

Request

```
POST https://secure.p03.eloqua.com/api/bulk/2.0/syncs

{
  "syncedInstanceUri" : "/contacts/exports/55",
  "callbackURL" : "https://www.example.com"
}
```

Response

```
{
   "syncedInstanceUri":"/contacts/exports/55",
   "callbackUrl":"http://www.example.com",
   "status":"pending",
   "createdAt":"2018-09-25T18:08:32.3485942Z",
   "createdBy":"API.User",
   "uri":"/syncs/66100"
}
```

The sync `status` will progress. Retrieve the sync to check the status of the sync.

Request

```
GET https://secure.p03.eloqua.com/api/bulk/2.0/sync/66100
```

Response

```
{
  "syncedInstanceURI": "/contacts/exports/55",
  "syncStartedAt": "2014-01-01T13:59:07.1375620Z",
  "syncEndedAt": "2014-01-01T13:59:14.1375620Z",
  "status": "success",
  "createdAt": "2014-01-01T13:59:07.1375620Z",
  "createdBy": "DocsExample",
  "uri": "/syncs/6"
}
```

After the sync has completed and the status is `success`, the last step is to retrieve the sync data.

Request

```
GET https://secure.p03.eloqua.com/api/bulk/2.0/syncs/66100/data
```

Response


```
{
  "totalResults": 1,
  "limit": 0,
  "offset": 0,
  "count": 0,
  "hasMore": true,
  "items": [
    {
      "email": "john.snow@example.com"
    }
  ]
}
```


Using the data

Your app will likely want to use the records for a specific purpose, after this is done, you can import the data into Eloqua.

Importing the data into Eloqua

Start by creating a bulk import definition, setting the `status` to `complete` to import data. For this example, we will demonstrate using a contact import definition, where our service is an action service and its instance GUID is `f82d50cd-86a9-4fca-b37e-4ec9a98b0339`.

If there is no data to import, and you only need to update a record's status, you can update a record's status without performing an import by creating a contact sync action definition.

> **Warning**: When referencing service instances, you must transmit the GUID without dashes. The bulk API will error if you transmit the GUID with the dashes.

Create a contact import definition.

Request

```
POST https: //secure.p03.eloqua.com/api/bulk/2.0/contacts/imports
{
  "name": "Awesome App Contact Import - f82d50cd-86a9-4fca-b37e-
4ec9a98b0339 - 12345",
  "updateRule": "always",
  "fields": {
    "emailAddress": "{{Contact.Field(C_EmailAddress)}}"
  },
  "syncActions": [
    {
      "destination": "{{ActionInstance
(f82d50cd86a94fcab37e4ec9a98b0339).Execution[12345]}}",
      "action": "setStatus",
      "status": "complete"
    }
  ],
  "identifierFieldName": "emailAddress"
}
```

Eloqua's response will be a 201 Created response that includes a `uri` parameter, which you can use to identify the import.

Response

```
{
  "name": "Awesome App Contact Import - f82d50cd-86a9-4fca-b37e-
4ec9a98b0339 - 12345",
  "updateRule": "always",
  "fields": {
    "emailAddress": "{{Contact.Field(C_EmailAddress)}}"
  },
  "identifierFieldName": "emailAddress",
  "syncActions": [
    {
      "destination": "{{ActionInstance
(f82d50cd86a94fcab37e4ec9a98b0339).Execution[12345]}}",
      "action": "setStatus",
      "status": "complete"
    }
  ],
  "isSyncTriggeredOnImport": false,
  "isUpdatingMultipleMatchedRecords": false,
  "uri": "/contacts/imports/6",
  "createdBy": "API.User",
  "createdAt": "2018-03-06T13:59:00.6600046Z",
```

```
  "updatedBy": "API.User",
  "updatedAt": "2018-03-06T13:59:00.6600046Z"
}
```

Send Eloqua the import data using the `uri`:

Request

```
POST https://secure.p03.eloqua.com/api/bulk/2.0/contacts/imports/6/data
[
  {
    "emailAddress": "john.snow@example.com"
  }
]
```

Eloqua's response will be a 204 No Content indicating that the data upload was successful.

App developer reference

The following pages provide detailed reference for the parameters and endpoints used in the Oracle Eloqua app Developer Framework.

- App level URL template parameters
- Service level URL template parameters
- App developer API endpoints
- Bulk API v2.0 documentation

❏**Warning**: For `PUT` or `POST` requests, you must specify `application/json` in the **Content-Type** header. If the **Content-Type** header is not included, data persisted in Eloqua will be corrupted, resulting in your App not functioning correctly.

The **Content-Type** header is not required for `GET` or `DELETE` requests, since these do not accept data inputs.

Service level URL template parameters

The Oracle Eloqua app Developer Framework supports *URL Templating*, enabling the configuration of service URIs Eloqua calls out to. Any acceptable *template parameter* is substituted with the appropriate value before the call is made. See the Introduction to URL templating for more information.

There are different *template parameters* available for different calls in different AppCloud service types:

- Action service parameters
- Decision service parameters
- Feeder service parameters

- Content service parameters
- Menu service parameters
- Firehose service parameters

To look up individual parameters, please see the below parameter descriptions.

Service parameters

Action services

URL	Available parameters	Conditional Parameters
Create URL	{InstanceId}, {InstallId}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {SiteId}, {AppId}, {EntityType}, {CustomObjectId}	
Configure URL	{InstanceId}, {InstallId}, {AssetType}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {SiteId}, {AppId}, {EntityType}, {CustomObjectId}	{AssetId} and {AssetName} are available **only if** the campaign has been saved with the referencing asset.
Copy URL	{InstanceId}, {InstallId},{AssetId}, {AssetName}, {AssetType}, {OriginalInstanceId}, {OriginalInstallId}, {OriginalAssetId}, {OriginalAssetName}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {SiteId}, {AppId}	
Notification URL	{InstanceId}, {InstallId},{AssetId}, {AssetName}, {AssetType}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {SiteId}, {AppId}, {ExecutionId}, {EntityType}, {CustomObjectId}	
Delete URL	{InstanceId}, {InstallId}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {SiteId}, {AppId}	

Decision services

URL	Available parameters	Conditional Parameters
Create URL	{InstanceId}, {InstallId}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {SiteId}, {AppId}, {EntityType}, {CustomObjectId}	
Configure URL	{InstanceId}, {InstallId}, {AssetType}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {SiteId}, {AppId}, {EntityType}, {CustomObjectId}	{AssetId} and {AssetName} are available **only if** the campaign has been saved with the referencing asset.
Copy URL	{InstanceId}, {InstallId},{AssetId}, {AssetName}, {AssetType}, {OriginalInstanceId}, {OriginalInstallId}, {OriginalAssetId}, {OriginalAssetName}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {SiteId}, {AppId}	
Notification URL	{InstanceId}, {InstallId},{AssetId}, {AssetName}, {AssetType}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {SiteId}, {AppId}, {ExecutionId}, {EntityType}, {CustomObjectId}	
Delete URL	{InstanceId}, {InstallId}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {SiteId}, {AppId}	

Feeder services

URL	Available parameters	Conditional Parameters
Create URL	{InstanceId}, {InstallId}, {UserName}, {UserId},	

URL	Available parameters	Conditional Parameters
	{UserCulture}, {SiteName}, {SiteId}, {AppId}, {EntityType}, {CustomObjectId}	
Configure URL	{InstanceId}, {InstallId}, {AssetType}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {SiteId}, {AppId}, {EntityType}, {CustomObjectId}	{AssetId} and {AssetName} are available **only if** the campaign has been saved with the referencing asset.
Copy URL	{InstanceId}, {InstallId},{AssetId}, {AssetName}, {AssetType}, {OriginalInstanceId}, {OriginalInstallId}, {OriginalAssetId}, {OriginalAssetName}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {SiteId}, {AppId}	
Notification URL	{InstanceId}, {InstallId},{AssetId}, {AssetName}, {AssetType}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {EventType}, {SiteId}, {AppId}, {EntityType}, {CustomObjectId}	
Delete URL	{InstanceId}, {InstallId}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {SiteId}, {AppId}	

Content services

URL	Available parameters	Conditional Parameters
Create URL	{InstanceId}, {InstallId}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {SiteId}, {AppId}	
Configure URL	{InstanceId}, {InstallId}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {SiteId}, {AppId}	{AssetId}, {AssetType} and {

URL	Available parameters	Conditional Parameters
		{AssetName} are available **only if** the email or landing page has been saved with the referencing asset.
Copy URL	{InstanceId}, {InstallId},{AssetId}, {AssetName}, {AssetType}, {OriginalInstanceId}, {OriginalInstallId}, {OriginalAssetId}, {OriginalAssetName}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {SiteId}, {AppId}	
Email Notification URL	{InstanceId}, {InstallId},{AssetId}, {AssetName}, {AssetType}, {UserName}, {UserId}, {SiteName}, {SiteId}, {AppId}, {RenderType}, {IsPreview}, {ExecutionId}	
Landing Page Notification URL	{InstanceId}, {InstallId},{AssetId}, {AssetName}, {AssetType}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {SiteId}, {AppId} ,{VisitorId}, {CampaignId}	
Delete URL	{InstanceId}, {InstallId}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {SiteId}, {AppId}	

Menu services

URL	Available parameters
Action URL	{InstallId}, {AssetId}, {AssetName}, {AssetType}, {AssetPath}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {SiteId}, {AppId}

Firehose services

URL	Available parameters
Notification URL	{InstallId},{AssetId}, {AssetName}, {assetType}, {UserName}, {UserId}, {UserCulture}, {SiteName}, {SiteId}, {AppId}, {EventType}

Parameter descriptions

- `InstanceId` is a templated parameter used in AppCloud service apps. Its value is the GUID for the specific instance of the AppCloud service being used. For instance, when a user drags and drops an action or decision service onto a campaign canvas this is considered a distinct instance and a new `InstanceId` is created.

- `InstallId` is a templated parameter used in AppCloud service apps. Its value is the GUID for the user's installation of the AppCloud App. Whenever a user installs an app, a new `InstallId` is created.

- `AssetId` is a templated parameter used in AppCloud service apps. Its value is the referencing asset's ID. Each instance of an asset will have a unique `AssetId`.

- `AssetName` is a templated parameter used in AppCloud service apps. Its value is the referencing asset's filename.

- `AssetType` is a templated parameter used in AppCloud service apps. Its value is the referencing asset's type. Possible asset types include campaign, form, landing page, program, etc.

- `AssetPath` is a templated parameter used in AppCloud service apps. Its value is the referencing asset's directory path.

- `OriginalInstanceId` is a templated parameter used in AppCloud service apps. It is specific to a service's Copy URL and its value is the original service's `InstanceId` (as opposed to the copied service's `InstanceId`).

- `OriginalInstallId` is a templated parameter used in AppCloud service apps. It is specific to a service's Copy URL and its value is the original service's `installId` (as opposed to the copied service's `InstallId`).

- `OriginalAssetId` is a templated parameter used in AppCloud service apps. It is specific to a service's Copy URL and its value is the original service's `AssetId` (as opposed to the copied service's `AssetId`).

- `OriginalAssetName` is a templated parameter used in AppCloud service apps. It is specific to a service's Copy URL and its value is the original service's `AssetName` (as opposed to the copied service's `AssetName`).

- `UserName` is a templated parameter used in AppCloud service apps. Its value is the name of the user who triggered the call.

- `UserId` is a templated parameter used in AppCloud service apps. Its value is the ID of the user who triggered the call.

- `UserCulture` is a templated parameter used in AppCloud service apps. Its value is the linguistic profile of the user of the user that triggered the call. Eloqua uses Microsoft's CultureInfo class to define culture. For example, for English (United States), the code is "en-US", for French (Canada) it is "fr-CA".

- `SiteName` is a templated parameter used in AppCloud service apps. Its value is the name of the user's Eloqua instance.

- `SiteId` is a templated parameter used in AppCloud service apps. Its value is the GUID-based ID for the user's Eloqua instance.

- `AppId` is a templated parameter used in AppCloud service apps. Its value is the GUID-based ID of the app making the service call. Each AppCloud app has a single unique `AppId`.

- `EventType` is a templated parameter used in AppCloud service apps. Its value is the status change that triggered the call (Created, Activated, Draft, etc.)

  List of `EventType` values: `Created`, `Updated`, `Deleted`, `Activated`, `ActivatedBySchedule`, `Scheduled`, `Draft`, `DraftWaitingApproval`, `DraftApproved`, `CampaignAssetAdded`, `CampaignAssetRemoved`, `CampaignLandingPageAdded`, `CampaignLandingPageRemoved`, `CampaignFormAdded`, `CampaignFormRemoved`, `CampaignEmailAdded`, `CampaignEmailRemoved`, `CampaignSegmentAdded`, `CampaignSegmentRemoved`, `MembersAdded`, `ContentRequired`, `Completed`.

- `VisitorId` is a templated parameter used in AppCloud content service apps. Its value is an integer and represents the ID of the visitor for whom you wish to construct a landing page.

- `RenderType` is a templated parameter used in AppCloud content service apps. Its value specifies the different types of email renderings. Possible values are:

    - `0`: No render -- no content is being assembled.
    - `1`: EmailSend -- content is being assembled for sending an email.
    - `2`: LiveWeb -- content is being assembled to render a live landing page.
    - `3`: EmailPreview -- content is being assembled to render an email preview.
    - `4`: WebPreview -- content is being assembled to render a preview of a landing page.
    - `5`: EmailSaved -- content is being assembled to render the web version of a sent email.

- `IsPreview` is a templated parameter used in AppCloud content service apps. Its value specifies whether the referencing email is in preview mode. Possible values are:
    - `0`: False
    - `1`: True

- `ExecutionId` is a templated parameter used in AppCloud content service apps. Its value is an integer and identifies a unique email send execution.

- `EntityType` is a templated parameter used in AppCloud service apps. The value specifies the entity type the campaign or program is set to. Possible values are `Contacts` or `CustomObjectRecords`.

  For instance, if a user drags a decision step onto a custom object program, the `EntityType` would be `CustomObjectRecords` because the program's entity type is set to custom objects. Alternatively, if that decision step was dragged onto a contact program, the `EntityType` would be `Contacts`.

  Note that campaigns only support the contact entity type, but programs can support contacts or custom objects. Use the AssetType parameter to determine if the asset is a campaign or a program.

- `CustomObjectId` is a templated parameter used in AppCloud action, decision, and feeder service apps for programs. Its value is the ID of the custom object associated with the program when `CustomObjectRecords` is specified for EntityType. It must be an integer.

- `CampaignId` is a templated parameter used in Landing Page Content services. Its value is the ID of the campaign retrieved using the `elqCampaignId` query parameter. It must be an integer. Learn more.

App level URL template parameters

The Oracle Eloqua appDeveloper Framework supports *URL Templating*, enabling the configuration of service URIs Eloqua calls out to. Any acceptable *template parameter* is substituted with the appropriate value before the call is made. See the Introduction to URL Templating page for more information.

To look up individual parameters, please see the below parameter descriptions.

App level parameters

Lifecycle parameters

URL	Description	Parameters
Enable URL	The URL called when the App is first installed or when it is reconfigured. This is a templated URL which should include, at a minimum, the {InstallId}, {AppId}, and {CallbackUrl} parameters. WIthout the callback URL you will not be able to complete the installation flow if your app requires configuring (for example, to use OAuth).	{InstallId}, {UserName}, {UserId}, UserCulture}, {SiteName}, {SiteId}, {AppId}, {CallbackUrl}
Configure URL	The URL called when clicking Configure (Gear icon next to Check Connection) for the App within the AppCloud Catalog.	{InstallId}, {UserName}, {UserId}, UserCulture}, {SiteName}, {SiteId}, {AppId}
Status URL	The URL called when a user checks the App's connection. Learn how to respond when Eloqua	{InstallId}, {UserName}, {UserId}, UserCulture},

URL	Description	Parameters
	calls the Status URL.	{SiteName}, {SiteId}, {AppId}
Uninstall URL	The URL called when a user uninstalls the app. This allows Eloqua to notify the app provider when someone uninstalls their app from the appcloud catalog. This URL supports templated parameters such as {InstallId}, {AppId}, etc.	{InstallId}, {UserName}, {UserId}, UserCulture}, {SiteName}, {SiteId}, {AppId}

OAuth parameters

URL	Description	Parameters
Callback URL	Your App's registered redirection endpoint. Its value should be the same as the `redirect_uri` used when requesting initial authorization for OAuth. Learn more about authentication with OAuth.	{InstallId}, {UserName}, {UserId}, UserCulture}, {SiteName}, {SiteId}, {AppId}

Parameter descriptions

- `InstanceId` is a templated parameter used in AppCloud service apps. Its value is the GUID for the specific instance of the AppCloud service being used. For instance, when a user drags and drops an action or decision service onto a campaign canvas this is considered a distinct instance and a new `InstanceId` is created.

- `InstallId` is a templated parameter used in AppCloud service apps. Its value is the GUID for the user's installation of the AppCloud App. Whenever a user installs an app, a new `InstallId` is created.

- `AssetId` is a templated parameter used in AppCloud service apps. Its value is the referencing asset's ID. Each instance of an asset will have a unique `AssetId`.

- `AssetName` is a templated parameter used in AppCloud service apps. Its value is the referencing asset's filename.

- `AssetType` is a templated parameter used in AppCloud service apps. Its value is the referencing asset's type. Possible asset types include campaign, form, landing page, program, etc.

- `AssetPath` is a templated parameter used in AppCloud service apps. Its value is the referencing asset's directory path.

- `CallbackURL` is the Eloqua URL an app calls into when configuration is complete during installation.**Example**:<base url>/Apps/Cloud/Admin/Install/Callback/{1}?guid={2}
  Where `{1}` is the installId and `{2}` is a guid generated by Eloqua. **Note**: this parameter is separate and distinct from the OAuth Callback URL.

- `OriginalInstanceId` is a templated parameter used in AppCloud service apps. It is specific to a service's Copy URL and its value is the original service's `InstanceId` (as opposed to the copied service's `InstanceId`).

- `OriginalInstallId` is a templated parameter used in AppCloud service apps. It is specific to a service's Copy URL and its value is the original service's `installId` (as opposed to the copied service's `InstallId`).

- `OriginalAssetId` is a templated parameter used in AppCloud service apps. It is specific to a service's Copy URL and its value is the original service's `AssetId` (as opposed to the copied service's `AssetId`).

- `OriginalAssetName` is a templated parameter used in AppCloud service apps. It is specific to a service's Copy URL and its value is the original service's `AssetName` (as opposed to the copied service's `AssetName`).

- `UserName` is a templated parameter used in AppCloud service apps. Its value is the name of the user who triggered the call.

- `UserId` is a templated parameter used in AppCloud service apps. Its value is the ID of the user who triggered the call.

- `UserCulture` is a templated parameter used in AppCloud service apps. Its value is the linguistic profile of the user of the user that triggered the call. Eloqua uses Microsoft's CultureInfo class to define culture. For example, for English (United States), the code is "en-US", for French (Canada) it is "fr-CA".

- `SiteName` is a templated parameter used in AppCloud service apps. Its value is the name of the user's Eloqua instance.

- `SiteId` is a templated parameter used in AppCloud service apps. Its value is the GUID-based ID for the user's Eloqua instance.

- `AppId` is a templated parameter used in AppCloud service apps. Its value is the GUID-based ID of the app making the service call. Each AppCloud app has a single unique `AppId`.

- `EventType` is a templated parameter used in AppCloud service apps. Its value is the status change that triggered the call (Created, Activated, Draft, etc.)

  List of `EventType` values: `Created`, `Updated`, `Deleted`, `Activated`, `ActivatedBySchedule`, `Scheduled`, `Draft`, `DraftWaitingApproval`, `DraftApproved`, `CampaignAssetAdded`, `CampaignAssetRemoved`, `CampaignLandingPageAdded`, `CampaignLandingPageRemoved`, `CampaignFormAdded`, `CampaignFormRemoved`, `CampaignEmailAdded`, `CampaignEmailRemoved`, `CampaignSegmentAdded`, `CampaignSegmentRemoved`, `MembersAdded`, `ContentRequired`, `Completed`.

- `VisitorId` is a templated parameter used in AppCloud content service apps. Its value is an integer and represents the ID of the visitor for whom you wish to construct a landing page.

- `RenderType` is a templated parameter used in AppCloud content service apps. Its value specifies the different types of email renderings. Possible values are:
  - `0`: No render -- no content is being assembled.
  - `1`: EmailSend -- content is being assembled for sending an email.
  - `2`: LiveWeb -- content is being assembled to render a live landing page.
  - `3`: EmailPreview -- content is being assembled to render an email preview.
  - `4`: WebPreview -- content is being assembled to render a preview of a landing page.
  - `5`: EmailSaved -- content is being assembled to render the web version of a sent email.

- `IsPreview` is a templated parameter used in AppCloud content service apps. Its value specifies whether the referencing email is in preview mode. Possible values are:
  - `0`: False
  - `1`: True

- `ExecutionId` is a templated parameter used in AppCloud content service apps. Its value is an integer and identifies a unique email send execution.

Eloqua app developer API endpoints

Cloud API endpoints

- PUT /api/cloud/1.0/actions/instances/{id}
- PUT /api/cloud/1.0/contents/instances/{id}
- PUT /api/cloud/1.0/decisions/instances/{id}
- PUT /api/cloud/1.0/feeders/instances/{id}
- POST /api/cloud/1.0/actions/instances

- POST /api/cloud/1.0/decisions/instances
- POST /api/cloud/1.0/feeders/instances
- GET /api/cloud/1.0/apps/{id}/configurations
- PUT /api/cloud/1.0/apps/{id}/configurations
- GET /api/cloud/1.0/apps/{id}/logs
- GET /api/cloud/1.0/apps/{id}/activeExecutions

Learn more

PUT `/api/cloud/1.0/actions/instances/{id}`

Updates an action service instance.

> **Warning**: For `PUT` requests, you must specify `application/json` in the **Content-Type** header. If the **Content-Type** header is not included, data persisted in Eloqua will be corrupted, resulting in your App not functioning correctly.

URL parameters

name	type	description
`id`	GUID	Unique identifier for that service instance

Request Body

name	type	description
`requiresConfiguration`	boolean	`requiresConfiguration` is an optional Boolean parameter which tells Eloqua whether user configuration is required before the app can be used. If set to `true`, users will be unable to activate

name	type	description
		a campaign or program containing the unconfigured app service instance. Eloqua will display an error message.
`recordDefinition`	Dictionary of strings	Defines the mapping between your fields and Eloqua's fields

Example

Update the record definition and configuration setting of the action service instance with GUID `7b95fe48-6598-43e8-8fd1-dcb40cf83ef6`:

```
PUT /api/cloud/1.0/actions/instances/7b95fe48-6598-43e8-8fd1-dcb40cf83ef6
{
   "recordDefinition": {
      "ContactID": "{{Contact.Id}}",
      "EmailAddress": "{{Contact.Field(C_EmailAddress)}}"
   },
   "requiresConfiguration": true
}
```

PUT `/api/cloud/1.0/contents/instances/{id}`

Updates a content service instance.

> ❏**Warning**: For `PUT` requests, you must specify `application/json` in the **Content-Type** header. If the **Content-Type** header is not included, data persisted in Eloqua will be corrupted, resulting in your App not functioning correctly.

URL parameters

name	type	description
`id`	GUID	Unique identifier for that service instance

Request body

name	type	description
`requiresConfiguration`	boolean	`requiresConfiguration` is an optional Boolean parameter which tells Eloqua whether user configuration is required before the app can be used. If set to `true`, users will be unable to activate a campaign or program containing the unconfigured app service instance. Eloqua will display an error message.
`recordDefinition`	Dictionary of strings	Defines the mapping between your fields and Eloqua's fields
`height`	Integer	Height of the image
`width`	Integer	Width of the image
`editorImageUrl`	URL	URL of the image to display in the Landing Page and Email editor

Example

Update the record definition of the content instance with GUID `7b95fe48-6598-43e8-8fd1-dcb40cf83ef6`:

```
PUT /api/cloud/1.0/contents/instances/7b95fe48-6598-43e8-8fd1-dcb40cf83ef6
{
  "recordDefinition": {
    "ContactID": "{{Contact.Id}}",
    "EmailAddress": "{{Contact.Field(C_EmailAddress)}}"
  },
  "height": 256,
  "width": 256,
  "editorImageUrl": "https://example.com/image.png"
}
```

PUT `/api/cloud/1.0/decisions/instances/{id}`

Updates a decision service instance.

❏**Warning**: For `PUT` requests, you must specify `application/json` in the **Content-Type** header. If the **Content-Type** header is not included, data persisted in Eloqua will be corrupted, resulting in your App not functioning correctly.

URL parameters

name	type	description
`id`	GUID	Unique identifier for that service instance

Request Body

name	type	description
`requiresConfiguration`	boolean	`requiresConfiguration` is an optional Boolean parameter which tells Eloqua whether user configuration is required before the app can be used. If set to `true`, users will be unable to activate a campaign or program containing the unconfigured app service instance. Eloqua will display an error message.
`recordDefinition`	Dictionary of strings	Defines the mapping between your fields and Eloqua's fields

Example

Update the record definition and configuration setting of the decision instance with GUID `7b95fe48-6598-43e8-8fd1-dcb40cf83ef6`:

```
PUT /api/cloud/1.0/decisions/instances/7b95fe48-6598-43e8-8fd1-dcb40cf83ef6

{
  "recordDefinition" :
    {
      "ContactID" : "{{Contact.Id}}",
      "EmailAddress" : "{{Contact.Field(C_EmailAddress)}}"
```

```
    }
   "requiresConfiguration": true
}
```

PUT `/api/cloud/1.0/feeders/instances/{id}`

Updates a feeder service instance.

> **Warning**: For `PUT` requests, you must specify `application/json` in the **Content-Type** header. If the **Content-Type** header is not included, data persisted in Eloqua will be corrupted, resulting in your App not functioning correctly.

URL parameters

name	type	description
`id`	GUID	Unique identifier for that service instance

Request Body

name	type	description
`requiresConfiguration`	boolean	`requiresConfiguration` is an optional Boolean parameter which tells Eloqua whether user configuration is required before the app can be used. If set to `true`, users will be unable to activate a campaign or program containing the unconfigured app service instance. Eloqua will display an error message.

Example

Update the feeder service instance with GUID `7b95fe48-6598-43e8-8fd1-dcb40cf83ef6`:

```
PUT /api/cloud/1.0/feeders/instances/7b95fe48-6598-43e8-8fd1-dcb40cf83ef6

{
  "requiresConfiguration": true
}
```

POST `/api/cloud/1.0/actions/instances`

Retrieves information for up to 200 action service instances, searched by service instance GUID.

> **Important**: Request header 'X-HTTP-Method-Override' needs to be set with value 'SEARCH'.

Request parameters

Name	Type	Description	Possible values
ids	array of GUIDs	The unique identifiers for the service instances to retrieve. Maximum of 200 GUIDs per request. • If a GUID does not exist, or has been deleted, the result will be excluded from the response. • If the array contains duplicate GUIDs, or exceeds the 200 limit, a 400 Bad Request will be returned. • The dashes within the GUID are required.	d7a34c94-c370-4958-8a88-b56d5621e68a

Response parameters

Name	Type	Description
recordDefinition	Dictionary of strings	The field mapping used in the service instance.
requiresConfiguration	boolean	Whether or not user configuration is required before the service can be used.
statement	string	The markup language statement used for referencing.
dependentAssetType	string	The asset type for which the service instance is being used. Possible values include: `Campaign` or `Program`. For example, if the service is on a campaign canvas, the `dependentAssetType` would be `Campaign`.
dependentAssetId	string	The asset Id for the asset for which the service instance is being used in.
uri	string	System-generated unique resource identifier.
createdBy	string	The login id of the user who created the service instance.
createdAt	string	The date and time the service instance was created, expressed in ISO 8601 date format.
updatedBy	string	The login id of the user that last updated the service instance.
updatedAt	string	The date and time the

Name	Type	Description
		service instance was last updated, expressed in ISO 8601 date format.

Example

Retrieve 2 action service instances:

```
POST api/cloud/1.0/actions/instances
X-HTTP-Method-Override: SEARCH
Content-Type: application/json
```

Request body:

```
{
 "ids": [
  "d7a34c94-c370-4958-8a88-b56d5621e68a",
  "1869b464-56f2-4334-923a-e631849d3cc8"
 ]
}
```

Response:

```
{
 "items": [
  {
   "recordDefinition": {
    "Id": "{{Contact.Id}}",
    "field2": "{{Contact.Field(C_FirstName)}}"
   },
   "requiresConfiguration": false,
   "statement": "{{ActionInstance(d7a34c94c37049588a88b56d5621e68a)}}",
   "dependentAssetType": "Campaign",
   "dependentAssetId": 49251,
   "uri": "/actions/618068/instances/d7a34c94c37049588a88b56d5621e68a",
   "createdBy": "Admin.User",
   "createdAt": "2017-03-03T15:31:32.8930000Z",
   "updatedBy": "Admin.User",
   "updatedAt": "2017-03-03T15:31:33.4000000Z"
```

```
    },
    {
      "recordDefinition": {
        "Email": "{{CustomObject[1].Field[4]}}"
      },
      "requiresConfiguration": false,
      "statement": "{{ActionInstance(1869b46456f24334923ae631849d3cc8)}}",
      "dependentAssetType": "Program",
      "dependentAssetId": 49252,
      "uri": "/actions/618068/instances/1869b46456f24334923ae631849d3cc8",
      "createdBy": "Admin.User",
      "createdAt": "2017-03-03T15:31:33.8930000Z",
      "updatedBy": "Admin.User",
      "updatedAt": "2017-03-03T15:31:34.2170000Z"
    }
  ]
}
```


POST `/api/cloud/1.0/decisions/instances`

Retrieves information for up to 200 decision service instances, searched by service instance GUID.

Important: Request header 'X-HTTP-Method-Override' needs to be set with value 'SEARCH'.

Request parameters

Name	Type	Description	Possible values
ids	array of GUIDs	The unique identifiers for the service instances to retrieve. Maximum of 200 GUIDs per request. • If a GUID does not exist, or has been deleted, the result will be excluded from the response.	c8b0a315-4bb1-412f-b6d1-9c82f91cd6c2

Name	Type	Description	Possible values
		• If the array contains duplicate GUIDs, or exceeds the 200 limit, a 400 Bad Request will be returned. • The dashes within the GUID are required.	

Response parameters

Name	Type	Description
recordDefinition	Dictionary of strings	The field mapping used in the service instance.
requiresConfiguration	boolean	Whether or not user configuration is required before the service can be used.
statement	string	The markup language statement used for referencing.
dependentAssetType	string	The asset type for which the service instance is being used. Possible values include: `Campaign` or `Program`. For example, if the service is on a campaign canvas, the `dependentAssetType` would be `Campaign`.
dependentAssetId	string	The asset Id for the asset for which the service instance is being used in.
uri	string	System-generated unique resource identifier.

Name	Type	Description
createdBy	string	The login id of the user who created the service instance.
createdAt	string	The date and time the service instance was created, expressed in ISO 8601 date format.
updatedBy	string	The login id of the user that last updated the service instance.
updatedAt	string	The date and time the service instance was last updated, expressed in ISO 8601 date format.

Example

Retrieve 2 decision service instances:

```
POST api/cloud/1.0/decisions/instances
X-HTTP-Method-Override: SEARCH
Content-Type: application/json
```

Request body:

```
{
  "ids": [
    "c8b0a315-4bb1-412f-b6d1-9c82f91cd6c2",
    "214e200f-7997-40fa-8329-5bdd847d7b60"
  ]
}
```

Important: Remember to include the dashes for the service instance GUIDs. A 400 validation error will be returned if the dashes are omitted.

Response:


```
{
  "items": [
    {
      "recordDefinition": {
        "Id": "{{Contact.Id}}",
        "field2": "{{Contact.Field(C_FirstName)}}"
      },
      "requiresConfiguration": false,
      "statement": "{{DecisionInstance(c8b0a3154bb1412fb6d19c82f91cd6c2)}}",
      "dependentAssetType": "Campaign",
      "dependentAssetId": 49254,
      "uri": "/decisions/618074/instances/c8b0a3154bb1412fb6d19c82f91cd6c2",
      "createdBy": "Admin.User",
      "createdAt": "2017-03-03T15:34:38.3470000Z",
      "updatedBy": "Admin.User",
      "updatedAt": "2017-03-03T15:34:38.9670000Z"
    },
    {
      "recordDefinition": {
        "Email": "{{CustomObject[1].Field[4]}}"
      },
      "requiresConfiguration": false,
      "statement": "{{DecisionInstance(214e200f799740fa83295bdd847d7b60)}}",
      "dependentAssetType": "Program",
      "dependentAssetId": 49255,
      "uri": "/decisions/618074/instances/214e200f799740fa83295bdd847d7b60",
      "createdBy": "Admin.User",
      "createdAt": "2017-03-03T15:34:39.3330000Z",
      "updatedBy": "Admin.User",
      "updatedAt": "2017-03-03T15:34:39.5930000Z"
    }
  ]
}
```


POST /api/cloud/1.0/feeders/instances

Retrieves information for up to 200 feeder service instances, searched by service instance GUID.

Important: Request header 'X-HTTP-Method-Override' needs to be set with value 'SEARCH'.

Request parameters

Name	Type	Description	Possible values
ids	array of GUIDs	The unique identifiers for the service instances to retrieve. Maximum of 200 GUIDs per request. • If a GUID does not exist, or has been deleted, the result will be excluded from the response. • If the array contains duplicate GUIDs, or exceeds the 200 limit, a 400 Bad Request will be returned. • The dashes within the GUID are required.	4de74ef1-bb2c-44d6-97ba-6acb1565bc58

Response parameters

Name	Type	Description
statement	string	The markup language statement used for referencing.
dependentAssetType	string	The asset type for which the service instance is being used. Possible values include: `Campaign` or `Program`. For example, if the service is on a campaign

Name	Type	Description
		canvas, the `dependentAssetType` would be `Campaign`.
dependentAssetId	string	The asset Id for the asset for which the service instance is being used in.
requiresConfiguration	boolean	Whether or not user configuration is required before the service can be used.
uri	string	System-generated unique resource identifier.
createdBy	string	The login id of the user who created the service instance.
createdAt	string	The date and time the service instance was created, expressed in ISO 8601 date format.
updatedBy	string	The login id of the user that last updated the service instance.
updatedAt	string	The date and time the service instance was last updated, expressed in ISO 8601 date format.

Example

Retrieve 2 feeder service instances:

```
POST api/cloud/1.0/feeders/instances
X-HTTP-Method-Override: SEARCH
Content-Type: application/json
```

Request body:

```
{
 "ids": [
  "4de74ef1-bb2c-44d6-97ba-6acb1565bc58",
  "97e3f90e-6ef8-40a8-9554-588a72de440e"
 ]
}
```


Important: Remember to include the dashes for the service instance GUIDs. A 400 validation error will be returned if the dashes are omitted.

Response:


```
{
 "items": [
  {
   "statement": "{{FeederInstance(4de74ef1bb2c44d697ba6acb1565bc58)}}",
   "dependentAssetType": "Campaign",
   "dependentAssetId": 6662,
   "requiresConfiguration": true,
   "uri": "/feeders/90289/instances/4de74ef1bb2c44d697ba6acb1565bc58",
   "createdBy": "API.User",
   "createdAt": "2018-06-01T17:57:00.1470000Z",
   "updatedBy": "API.User",
   "updatedAt": "2018-06-01T17:57:23.5130000Z"
  },
  {
   "statement": "{{FeederInstance(97e3f90e6ef840a89554588a72de440e)}}",
   "dependentAssetType": "Program",
   "dependentAssetId": 1385,
   "requiresConfiguration": true,
   "uri": "/feeders/90289/instances/97e3f90e6ef840a89554588a72de440e",
   "createdBy": "API.User",
   "createdAt": "2018-06-01T18:10:21.5700000Z",
   "updatedBy": "API.User",
   "updatedAt": "2018-06-01T18:10:36.8770000Z"
  }
 ]
}
```


GET `/api/cloud/1.0/apps/{id}/configurations`

Retrieves an Oracle Eloqua app App configuration.

URL parameters

name	type	description
`id`	GUID	The AppId of the app configuration you are retrieving

Request body

None.

Example

Example: Retrieve the App configuration file for the App with GUID `7b95fe48-6598-43e8-8fd1-dcb40cf83ef6`

```
GET /api/cloud/1.0/apps/7b95fe48-6598-43e8-8fd1-dcb40cf83ef6/configurations
```

PUT `/api/cloud/1.0/apps/{id}/configurations`

Updates an Oracle Eloqua app App configuration.

Warning: changes to the configuration file directly affect the app. Accidentally changing the `enableUrl`, for example, will result in people being unable to install your app. Ensure that any changes you push out are intentional.

URL parameters

name	type	description
`id`	GUID	The AppId of the app configuration you are updating

Request Body

name	type	description
`requiresConfiguration`	boolean	`requiresConfiguration` is an optional Boolean parameter which tells Eloqua whether user configuration is required before the app can be used. If set to `true`, users will be unable to activate a campaign or program containing the unconfigured app service instance. Eloqua will display an error message.
`uri`	URI	specific URI that describes the app
`clientSecret`	token	Eloqua-generated OAuth token
`name`	string	name of the app
`description`	string	description of the app
`shortDescription`	string	a brief description of the app
`iconUrl`	URL	URL pointing to the icon that Eloqua should display for the app
`enableUrl`	templated URL	URL called when the App is first installed or when it is reconfigured
`statusUrl`	templated URL	URL called when a user checks the app's status
`callbackUrl`	templated URL	URL that the users should be redirected to after installing an app
`makeAvailableUrl`	URL	Eloqua-generated URL used to add the app to a user's catalogue
`publishedSites`	dictionary of strings	lists the sites where the app is installed
`status`	string	Describes the status of the App - either "Up", "Down", or "Maintenance".

Example

Example: Set the status for the App with GUID `7b95fe48-6598-43e8-8fd1-dcb40cf83ef6` to **Down**.

```
PUT /api/cloud/1.0/apps/7b95fe48-6598-43e8-8fd1-dcb40cf83ef6/configurations
{
   "uri": "/apps/a0f6779e-8263-4bfb-a292-9714cd10d1e7/configurations",
   "clientSecret":
"a1ba63f9aae1cc84d91544f15b0af27ab6ced4e42c6145b9cc0425742ecf2a0da1ba63f
9aae1cc84d91544f15b0af27ab6ce",
   "name": "Call Example",
   "description": "This is a test",
   "shortDescription": "This is a test",
   "iconUrl": "https://www.example.com/logo.png",
   "enableUrl": "http://example.com/enable/{appId}/{installId}/{userName}/{siteName}/
{siteId}",
   "statusUrl": "http://example.com/status/{installId}",
   "callbackUrl": "http://example.com/callback/{installId}",
   "makeAvailableUrl": "https://example.com",
   "publishedSites": [
      "AwesomeCompany"
   ],
   "status": "Down"
}
```

GET `/api/cloud/1.0/apps/{id}/logs`

Retrieves an app's outbound logs. This endpoint can be used to retrieve logs for apps across different pods, while the logs page in the Eloqua interface is limited only to displaying outbound requests for your instance and other clients on your pod.

This endpoint orders results by the request date, where the latest outbound requests are ordered first. The outbound logs returned in the request will differ depending on the authentication used to access the endpoint.

If authenticated using oAuth:

- The outbound logs for a provider's apps are displayed. Attempting to access logs for other apps will return a 403 forbidden response code.

If authenticated using basic authentication:

- You can access logs for each app within your instance.

URL parameters

name	type	description
`id`	GUID	The AppId of the app log you are retrieving.
`serviceInstanceId`	GUID	The service instance Id of the app you are retrieving.
`startDate`	dateTime	The start date to filter outbound logs, expressed in ISO 8601 date format.
`endDate`	dateTime	The end date to filter outbound logs, expressed in ISO 8601 date format.
`limit`	integer	A URL parameter that specifies the maximum number of records to return.
`offset`	integer	Specifies an offset that allows to retrieve the next batch of records.
`totalResults`	boolean	A URL parameter that captures the total number of records that satisfy the request. This is not the count returned in the current response, but total count on the server side.

Request Body

None.

Example

Example: Retrieve the outbound logs for the App configuration with GUID `7b95fe48-6598-43e8-8fd1-dcb40cf83ef6`

```
GET /api/cloud/1.0/apps/7b95fe48-6598-43e8-8fd1-dcb40cf83ef6/logs
```

GET /api/cloud/1.0/apps/{id}/activeExecutions

Retrieves an app's active excutions by service instance id.

This endpoint orders results ascending by the execution id.

URL parameters

(missing or bad snippet)

name	type	description
`id`	GUID	The AppId of the app active executions you are retrieving.
`serviceInstanceId`	GUID	The service instance id of the app active executions you are retrieving.
`startDate`	dateTime	The start date to filter active executions, expressed in ISO 8601 date format.

(missing or bad snippet)

name	type	description
`endDate`	dateTime	The end date to filter active executions, expressed in ISO 8601 date format.
`limit`	integer	A URL parameter that specifies the maximum number of records to return. The default and maximum is 1000.
`offset`	integer	Specifies an offset to retrieve the next batch of records.

Request Body

None.

Example

Example: Retrieve the active executions for the App with GUID `920ecfda-7cd5-4c71-8714-8d35e3c3379d`, the service instance with GUID `9dbd4a8d-10c1-4dfd-8158-232dc34f21c8`, and a start date of October 27, 2025 10:00 AM

```
GET /api/cloud/1.0/apps/920ecfda-7cd5-4c71-8714-8d35e3c3379d/activeExecutions?startDate=2025-10-27T10:00&serviceInstanceId=9dbd4a8d-10c1-4dfd-8158-232dc34f21c8
```

App developer frequently asked questions

Why should I develop apps for the app developer framework?

The app developer framework is a complete development platform where you can easily build, register, and deploy apps for Eloqua. With new and improved service types which take advantage of Eloqua's bulk API, support for OAuth, and the ability to test your applications with Eloqua prior to launch, the App Developer Framework provides the environment needed to put apps first.

What permissions do I need to start developing apps for the Oracle Eloqua app?

At the minimum, you need access to an Eloqua instance. If you are not currently an Eloqua user, you can sign up as a technology partner to obtain a development instance. As a user, you will also need the **Advanced Users - Marketing** permissions.

When and how does my app get listed on the Oracle Eloqua app site?

See building apps for the Oracle marketing app developer framework on Topliners for detailed instructions.

How do marketers find my app to start using it?

Registered apps are listed on the Oracle Cloud Marketplace. Users are linked to this page through the "Get more Apps" link in Eloqua's **AppCloud Catalog** section.

No. The Oracle Eloqua app exclusively supports contact, account, activity and custom object fields in its record definition fields. You must use the correct Eloqua markup language statement to reference each field.

Is there an Oracle Eloqua app certification program?

Yes. Check out the Oracle Marketing AppCloud Certification Program on Topliners.

What kind of digital certificate is required to communicate with Eloqua?

App's URL endpoints must have an SSL Certificate, specifically needing to be a digital signature from a certificate authority (CA). Eloqua will not communicate with untrusted sites.

What happens during an Eloqua release?

The application and the various associated services will be intermittently unavailable within the duration of the maintenance window. If there are any Bulk API sync failures, they should be retried after the maintenance window completes.

Which version of TLS is supported?

TLS 1.2 is supported.

Oracle Eloqua App Services and Operations

Which service should I develop?

It depends on what you're trying to achieve! See the Service Descriptions for an overview of each service and the use cases it supports.

What's the difference between Oracle Eloqua App services and Cloud Connectors or Components?

Eloqua Oracle Eloqua app services greatly extend the functionality provided by cloud connectors and cloud components. Cloud content replaces cloud components, allowing you to process emails in bulk and test the service within Eloqua. Unlike cloud connectors and components, the Oracle Eloqua App services use Eloqua's bulk API for processing, greatly improving performance and throughput.

Can I include campaign, email, landing page, form fields in my record definition?

No. The Oracle Eloqua app exclusively supports contact, account, activity and custom object fields in its record definition fields. You must use the correct Eloqua markup language statement to reference each field.

Can I include static values in my record definition?

No. You can only specify the Eloqua markup language for Eloqua fields.

Managing Your Apps

What does this app status mean?

See the Oracle Eloqua Help Center for a list of all the different status messages you can see related to app members moving through your campaigns and programs.

What if my request to refresh a token times out?

If the current access token has not been used, submitting a request to authenticate with the previous refresh token will return the existing new access token and refresh token.

What happens if contacts encounter an error in an Action or Decision step? What happens if the service is unavailable when contacts flow into a step on the canvas?

The contacts remain in the step until the marketer manually pushes them into the next step on the canvas.

Does Eloqua notify me when someone uninstalls or deletes my app from their Eloqua instance?

If you set an Uninstall URL it will be called when a user uninstalls the app.

Why has my app been shut down?

If in the last five minutes, there were more than 100 calls, and 25% of them failed, the app will be shut down. All the pending and future requests will be marked as failed. See the App shutdown for more information.

Will Eloqua retry my contacts if my app doesn't respond?

No, the contacts in a step will be marked as "Errored". If the marketer has configured a default path for the contacts to flow through, then the contacts will flow into the next step.

What if my app responds to the notify call after 100 seconds?

Any response after 100 seconds will be ignored.

What if my app responds to the notify call with a response status code that is not 200 level?

If Eloqua calls out to your app and receives a response status code that is between 300 and 599, Eloqua will retry the notify call over approximately an eight-hour period of time with a backoff strategy of the time between calls doubling after each call. After this eight-hour period, the contacts in a step will be marked as "Errored". If the marketer has configured a default path for the contacts to flow through, then the contacts will flow into the next step.

Why does my app prompt the error "502 Error: App Provider not Available"?

This error message will appear if a URL has been configured with a non-standard port. Eloqua only supports the standard ports 80 and 443. If a URL is not configured with port 80 or port 443, requests to this URL will fail.

What happens to notifications when an app is set to down?

If an app is set to down for more than 24 hours Eloqua will no longer retry or save notifications. The app will need to be set back to Up in order to receive new notifications.

What happens if you try to send an email via the email deployment Application API endpoint that includes a Content Service with an app that is in a Reinstall state in the App catalog?

A 400 level error will be returned if trying to send an email via the email deployment Application API endpoint that includes a Content Service with an app that is in a Reinstall state.

Limits

Are there limits that I should be aware of?

The AppCloud developer framework relies on the bulk API. The bulk API has limits on the size of the staging area for imports and exports, on the amount of data you can import at one time, and on the number of fields you can include in a record definition. There is also a daily limit on the number of syncs you can perform.

What happens if I reach the daily sync limits?

The daily sync limit is not currently enforced, but syncs are logged and monitored.