Google Ads

Migrating from Feed-based to Asset-based Extensions

Google Ads API Migration Workshops - 2021

In this session, we will build a tool that migrates one Feed-based extension to an Asset-based extension. The new Asset-based extension will copy all of the Feed-based extension's settings and associations. This migration is relevant to all users that manage Feed-based extensions in Google Ads accounts.

"Feed-based" refers to any extensions managed through Extension Setting Services or Feeds Services. Asset-based extensions replace Feed-based extensions, and you'll need to migrate your existing extensions to Assets to continue maintaining them. The underlying Feed infrastructure is being retired; see our extension migration guide page for the sunset schedule. This workshop steps through the migration procedure described in our documentation. Please check the extension migration guide page for language-specific examples.

We will be working with the Google Ads API exclusively in this workshop, as Asset-based extensions are not supported in the AdWords API. The included code snippets are in C#, but the workflow, objects, and methods utilized apply to all Google Ads API client libraries. Note that we use `PascalCase` notation in this document, which follows the C# convention; please substitute `snake_case` or `camelCase` as fits your language of choice.

This session is divided into six parts:

- Install a Google Ads API client library and create a skeleton program
- Identify a Feed-based extension to migrate
- Fetch the Feed-based extension's contents
- Create an Asset-based extension
- Associate the Asset-based extension
- Remove the Feed-based extension

This workshop will demonstrate migrating an extension managed through Extension Setting services. The same general steps apply if you are managing your extensions through Feed services. However, you will need to change the types and reports mentioned. For example, use `CampaignFeedService` instead of `CampaignExtensionSettingService`, and the `Feed` type instead of `ExtensionFeedItem`.

This is meant to be an interactive session in which you can follow along with the demonstration by performing each of the steps below. Please post any questions you have to the Q&A forum, and our team will be standing by to help you out.

Part 1 - Prerequisites

You will need to have a Google Ads API client library installed to complete this workshop. Our goal in this part is to have a skeleton program to work with in this workshop. If you've already installed a client library and can successfully launch one of the example programs, proceed to Step 2.

Part 1.1: Install a client library

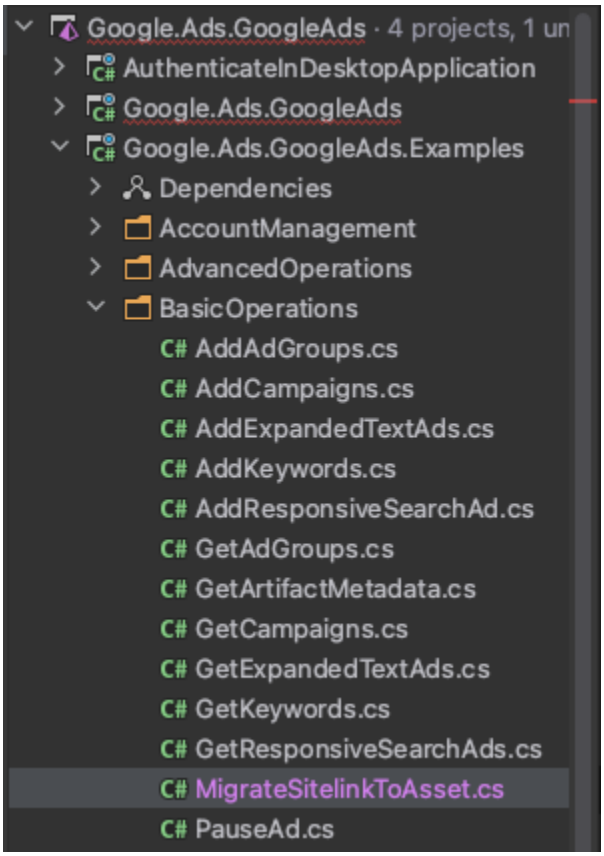
Visit the Google Ads Client Libraries overview page. Select one of the supported languages on the left side of the page and follow the instructions to install and configure the client library. Try to run one of the examples, such as `GetCampaigns` in the `BasicOperations` folder, to verify that your configuration is working correctly.

We'll be working with Sitelink extensions in this workshop. The Sitelink Asset-based extension is only supported in Google Ads API v8 or later; please upgrade if you are currently using any other version.

Part 1.2: Create a skeleton program

We'll write a standalone utility in this workshop that migrates one Sitelink extension. Navigate to your client library's `BasicOperations` folder in the examples directory. We'll use the existing `GetCampaigns` example as a base for our new program. Make a copy of the `GetCampaigns` file, and rename it to `MigrateSitelinkToAsset`. Your `BasicOperations` directory should look like the following:

1.2.0: Examples/BasicOperations directory after copying and renaming the file



Open the `MigrateSitelinkToAsset` file and make the following changes:

1. Change any occurrences of "`GetCampaigns`" to "`MigrateSitelinkToAsset`".
2. Remove the contents of the runner function. Note that this function is different in each library; it's the function named `Run` in our C# library, `runExample` in the Java and PHP libraries, `main` in the Python library, and the function whose name matches the file name (`migrate_sitelink_to_asset`) in the Perl and Ruby libraries.

When done, you should have a program that accepts the `CustomerId` argument and that has an empty runner function.

Part 1.3: Create dummy extensions (optional)

We'll be making changes to an active Sitelink extension in your account. We recognize that you might prefer to not modify the behavior of active ads, or might not have existing Sitelink extensions to work with. Fortunately, there are two example programs included in your client library that will create mock resources for you to utilize in this workshop. You might also want to use a test account to ensure that no changes are made to your current ads.

Run the `AddCampaign` example in the `BasicOperations` directory to create paused mock campaigns. The example will create several campaigns and print their resource names, which have the form `customers/{customerId}/campaigns/{campaignId}`.

Run the `AddSitelinks` example in your client library's `Extensions` directory to create mock Feed-based Sitelink extensions. This example will create four Sitelink extensions with various dates and targeting and attach them to a specified campaign. If you generated campaigns using the `AddCampaign` example, pass one of the printed `campaignId` values.

Part 2 - Identify the Feed to migrate

We'll be using Google Ads Query Language (GAQL) queries to get information from Google Ads API. GAQL queries return actual objects with their current values, making it easy to transform and update resources in your account.

We'll need to issue a search request for each query. We'll start by creating a GAQL query to fetch the IDs of the active Feed-based Sitelink extensions currently active in your account.

Begin by instantiating a `GoogleAdsServiceClient`:

2.0.0: Instantiate a `GoogleAdsServiceClient`

```
GoogleAdsServiceClient googleAdsService =
  client.GetService(Services.V8.GoogleAdsService);
```

Next, store the GAQL query in a string variable. Below is the minimum query, but you can add additional fields to narrow the results:

2.0.1: GAQL query for the IDs of active Sitelink extension feed items

```
string query = @"
  SELECT extension_feed_item.id
  FROM extension_feed_item
  WHERE
    extension_feed_item.status = 'ENABLED'
    AND extension_feed_item.extension_type = 'SITELINK'";
```

Lastly, we'll issue our query using a streaming search and iterate through the results. Note that the suggested implementation for searches can differ greatly by client library language; look at the model provided in the `BasicOperations/GetCampaigns` example in your client library:

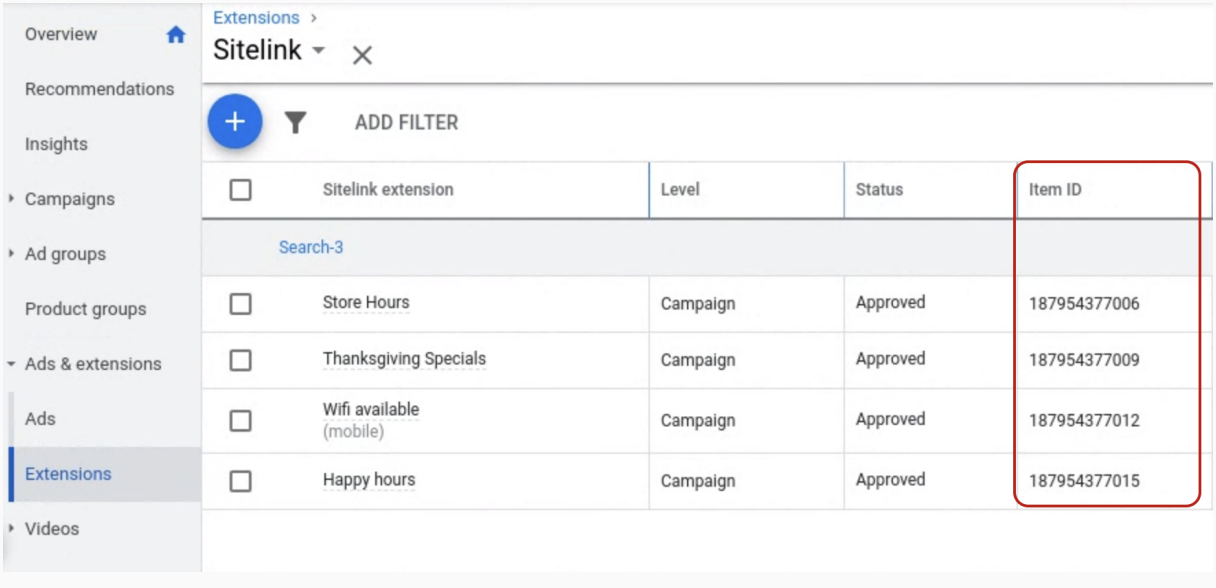
2.0.2: Issue a streaming search request and print the extension feed item IDs

```
googleAdsService.SearchStream(customerId.ToString(), query,
  delegate (SearchGoogleAdsStreamResponse response)
  {
    foreach (GoogleAdsRow googleAdsRow in response.Results)
    {
      Console.WriteLine("ExtensionFeedItem with ID {0} was found.",
        googleAdsRow.ExtensionFeedItem.Id);
    }
  }
);
```

Run your program, and any matching Extension Feed Item IDs will be printed to your console.

You can also find your Feed Item IDs though the Google Ads UI. To do so, go to the Extensions page and choose an extension type, such as Sitelink. Click **Columns → Modify columns**, then check the **Item ID** box under **Attributes**. Click **Apply**, and the ID will appear in the table view:

2.0.3: Example Google Ads UI extensions list with Item ID column



Pick one Item ID to work with before continuing. Either create a variable `feedItemId` in your program and assign it the chosen Item ID, or modify the program to accept a `feedItemId` argument and pass in the chosen Item ID when you run the program in later steps.

Part 3 - Fetch the extension contents

Part 3.1: Get the ExtensionFeedItem values

Our goal at the end of this part is to have collected all of the stored values for one Feed-based Sitelink extension. We'll do so by issuing a GAQL query to fetch the extension's contents. Several fields are common to all extension types, but you'll need to also request the fields for the particular extension type. The example below includes the Sitelink fields; use the Google Ads Query Builder to help identify relevant fields for other extension types.

Responses from `Search` and `SearchStream` contain fully-constructed instances of the objects requested. Using the `feedItemId` retrieved in the previous step, request and store the `ExtensionFeedItem` from the search response:

3.1.0: Request and store the contents of a Sitelink extension feed item

```
long feedItemId = /* FEED ITEM ID FROM PART 2 */;

string extensionFeedItemQuery = $@"
  SELECT
    extension_feed_item.ad_schedules,
    extension_feed_item.device,
    extension_feed_item.start_date_time,
    extension_feed_item.end_date_time,
    extension_feed_item.id,
    extension_feed_item.status,
    extension_feed_item.sitelink_feed_item.final_mobile_urls,
    extension_feed_item.sitelink_feed_item.final_url_suffix,
    extension_feed_item.sitelink_feed_item.final_urls,
    extension_feed_item.sitelink_feed_item.line1,
    extension_feed_item.sitelink_feed_item.line2,
    extension_feed_item.sitelink_feed_item.link_text,
    extension_feed_item.sitelink_feed_item.tracking_url_template
  FROM extension_feed_item
  WHERE extension_feed_item.extension_type = 'SITELINK'
```

```
    AND extension_feed_item.id = {feedItemId}
  LIMIT 1";

ExtensionFeedItem fetchedExtensionFeedItem = null;

// Issue a search request to get the extension feed item contents.
googleAdsService.SearchStream(customerId.ToString(),
  extensionFeedItemQuery,
  delegate (SearchGoogleAdsStreamResponse response)
  {
    fetchedExtensionFeedItem = response.Results.First().ExtensionFeedItem;
  }
);
```

Note that we've created a variable `fetchedExtensionFeedItem` that points to the returned `ExtensionFeedItem` object. In the next steps, we'll extract information from this object and use it to build our new `Asset`.

Part 3.2: Get any URL Custom Parameters (optional)

You can skip this step if you don't use URL Custom Parameters with your extensions.

URL Customer Parameters are not available from the `extension_feed_item` report used in Step 1; we'll have to fetch them using the `feed_item` report and add them to `fetchedExtensionFeedItem` manually. The procedure is very similar to the previous step:

3.2.0: Request any URL Custom Parameters and add them to an extension feed item

```
string urlCustomParametersQuery = $@"
  SELECT feed_item.url_custom_parameters
  FROM feed_item
  WHERE feed_item.id = {feedItemId}";

googleAdsService.SearchStream(customerId.ToString(),
  urlCustomParametersQuery,
```

```
  delegate (SearchGoogleAdsStreamResponse response)
  {
    RepeatedField<CustomParameter> urlCustomParameters =
      response.Results.First().FeedItem.UrlCustomParameters;

    if (urlCustomParameters.Count > 0)
    {
      fetchedExtensionFeedItem.SitelinkFeedItem.UrlCustomParameters.Add(
        urlCustomParameters);
      }
  }
);
```

At this point, your `ExtensionFeedItem` instance should be fully populated, and we are now ready to create the `Asset` object that will become the core of our Asset-based extension.

Part 4 - Create an Asset-based extension

Part 4.1: Create a new Asset instance

We'll now construct our `Asset` object and upload it to our account. Start by creating an `Asset` object and populating its top-level fields. These top-level fields are common to all assets, such as its name and URL settings. We'll copy these values from the `ExtensionFeedItem` retrieved in the previous step.

For simplicity, let's create a variable `sitelinkFeedItem` that refers to the Sitelink-specific contents of the `ExtensionFeedItem`:

4.1.0: Create a variable referring to an extension feed item's Sitelink-specific contents

```
SitelinkFeedItem sitelinkFeedItem = fetchedExtensionFeedItem.SitelinkFeedItem;
```

Then, set the `Asset`'s top-level fields. The scalar fields can be assigned directly, but you may need to append any URL Custom Parameters and Final URLs to the `Asset`'s repeated fields separately, depending on your client library language:

4.1.1: Create an `Asset` instance and populate its fields

```
Asset asset = new Asset
{
    // Name field is optional.
    Name = $"Migrated from feed item #{fetchedExtensionFeedItem.Id}",
    TrackingUrlTemplate = sitelinkFeedItem.TrackingUrlTemplate,
    FinalUrlSuffix = sitelinkFeedItem.FinalUrlSuffix
};

asset.FinalUrls.Add(sitelinkFeedItem.FinalUrls);
asset.FinalMobileUrls.Add(sitelinkFeedItem.FinalMobileUrls);
asset.UrlCustomParameters.Add(sitelinkFeedItem.UrlCustomParameters);
```

Part 4.2: Construct the SitelinkAsset instance

Next, we'll copy the Sitelink-specific fields from our `ExtensionFeedItem` to our new `Asset`. We'll do so by creating and populating a `SitelinkAsset` instance and attaching it to the `Asset` instance created in the previous step. Note that `Assets` can represent a variety of different types of data in addition to extensions, such as images and text; we set the `Asset`'s type by attaching an instance of one of these subtypes.

First, set the `Asset`'s `SitelinkAsset` field to a new `SitelinkAsset` instance. Then populate its fields from the `fetchedExtensionFeedItem`. Again, note that you may need to append values to the `AdScheduleTargets` repeated field separately depending on your client library language:

4.2.0: Create a `SitelinkAsset` instance and populate its fields

```
asset.SitelinkAsset = new SitelinkAsset
{
  Description1 = sitelinkFeedItem.Line1,
  Description2 = sitelinkFeedItem.Line2,
  LinkText = sitelinkFeedItem.LinkText,
};

asset.SitelinkAsset.AdScheduleTargets.Add(fetchedExtensionFeedItem.AdSchedules);
```

Note that some `Asset` fields may expect slightly different values than were used in Feeds. For example, the `PromotionAsset.percent_off` uses a different ratio than `PromotionFeedItem.percent_off`. Check the documentation for the particular extension type to see if any conversions are necessary. In the case of Sitelinks, no conversions are necessary.

Lastly, we'll set the extension's start and end dates. These dates should be strings in yyyy-MM-dd format; this is slightly different from `ExtensionFeedItem`'s YYYY-MM-DD HH:MM:SS format. Unlike in `ExtensionFeedItems`, these dates are set at the `Asset` subtype level. Note that an extension may not necessarily have a start or end date set; leave the start date unset to an immediately allow the extension to serve, and leave the end date unset for the extension to serve indefinitely:

4.2.1: Assign start and end dates to the `SitelinkAsset`

```
// Check if the StartDateTime field is populated.
if (fetchedExtensionFeedItem.HasStartDateTime)
{
  asset.SitelinkAsset.StartDate =
    DateTime.Parse(fetchedExtensionFeedItem.StartDateTime)
      .ToString("yyyy-MM-dd");
}

// Check if the EndDateTime field is populated.
if (fetchedExtensionFeedItem.HasEndDateTime)
{
  asset.SitelinkAsset.EndDate =
    DateTime.Parse(fetchedExtensionFeedItem.EndDateTime)
      .ToString("yyyy-MM-dd");
}
```

Part 4.3: Upload the Asset to Google Ads

Create instances of `AssetServiceClient` and `AssetOperation`. Set the `AssetOperation`'s `Create` field to the newly created `Asset`:

4.3.0: Instantiate an `AssetAdsServiceClient` and an `AssetOperation`

```
AssetServiceClient assetServiceClient =
  client.GetService(Services.V8.AssetService);

AssetOperation operation = new AssetOperation
{
  Create = asset
```

```
};
```

Then issue a mutate request containing the `AssetOperation`. Be sure to store the resource name returned in the response object; we'll need it for the next step:

4.3.1: Send a mutate request to create the Asset and save the resulting resource name

```
MutateAssetsResponse response = assetServiceClient.MutateAssets(
  customerId.ToString(), new[] { operation });
Console.WriteLine("Created Sitelink asset with resource name " +
  $"{response.Results.First().ResourceName}");
string assetResourceName = response.Results.First().ResourceName;
```

We'll use the `assetResourceName` in the next part to attach the `Asset` to the same campaigns and ad groups as the original Feed-based extension.

Part 5 - Associate the Asset-based extension

In this part, we'll attach the new `Asset` to the same accounts, campaigns, or ad groups as the original Feed-based extension. To do so, we'll need to determine which resources the Feed-based extension was associated with, then make new associations for our `Asset`.

Part 5.1: Identify resources associated with the Feed-based extension

First, we'll need to issue another GAQL query to request the relevant IDs and their attached extension feed items. We'll issue the query and build a list of IDs that the Feed-based extension was associated with. We cannot request only the extension settings for a specific Feed; instead, we'll need to request all of the extension settings for a particular resource type and filter those that contain our Feed-based extension:

5.1.0: Issue a streaming search request and store extension settings and campaign IDs

```
string extensionFeedResourceName =
  ResourceNames.ExtensionFeedItem(customerId, feedItemId);

List<CampaignExtensionSetting> campaignExtensionSettings =
  new List<CampaignExtensionSetting>();
List<long> campaignIds = new List<long>();

string campaignExtensionsQuery = @"
  SELECT campaign.id,
    campaign_extension_setting.extension_feed_items,
    campaign_extension_setting.resource_name
  FROM campaign_extension_setting
  WHERE campaign_extension_setting.extension_type = 'SITELINK'
    AND campaign.status != 'REMOVED'";

googleAdsService.SearchStream(customerId.ToString(), campaignExtensionsQuery,
  delegate(SearchGoogleAdsStreamResponse response)
  {
    foreach (GoogleAdsRow googleAdsRow in response.Results)
    {
```

```
      if (googleAdsRow.CampaignExtensionSetting.ExtensionFeedItems.Contains(
        extensionFeedResourceName))
      {
        Console.WriteLine(
          $"Found matching campaign with ID {googleAdsRow.Campaign.Id}.");
        campaignIds.Add(googleAdsRow.Campaign.Id);
        campaignExtensionSettings.Add(googleAdsRow.CampaignExtensionSetting);
      }
    }
  }
);
```

You may notice that we also store the returned details of any `ExtensionSetting` that contains the target `ExtensionFeedItem`. This is for later use in Part 6.

Note that we're only demonstrating fetching the *campaigns* that the extension is associated with in the above snippet; the same procedure applies to *ad group* and *account* level associations; simply substitute `AdGroup` and `Customer`, respectively, in place of `Campaign`. For example, use this query to get the ad groups that the Feed-based extension is associated with:

5.1.1: GAQL query for ad group extension settings

```
string adGroupsExtensionsQuery = @"
  SELECT ad_group.id,
    ad_group_extension_setting.extension_feed_items,
    ad_group_extension_setting.resource_name
  FROM ad_group_extension_setting
  WHERE ad_group_extension_setting.extension_type = 'SITELINK'
    AND ad_group.status != 'REMOVED'";
```

Likewise, change references as needed in the handling of the results, such as `CampaignExtensionSetting` → `AdGroupExtensionSetting` and `googleAdsRow.Campaign.Id` → `googleAdsRow.AdGroup.Id`. Again save the results in a list of IDs.

Proceed to the next step once you have all the collections of IDs that you'd like to work with.

Part 5.2: Link the Asset-based extension to the resources

We'll now use the collected lists of IDs to create and upload instances of `AdGroupAsset`, `CampaignAsset`, and `CustomerAsset` objects. These represent associations between one resource and one Asset.

We'll again demonstrate how to create and upload `CampaignAssets`, but the same procedure applies to ad groups and accounts; simply substitute `AdGroup` or `Customer` in the snippets below.

Begin by creating a new collection of `CampaignAssetOperations`. We'll fill this collection with one `CampaignAssetOperation` for each campaign to associate with our new `Asset`. For each campaign ID fetched in the previous step, create an instance of `CampaignAssetOperation` and set its `create` field to a new instance of `CampaignAsset`. Set the `CampaignAsset's asset` and `campaign` fields to the resource names of the `Asset` and campaign, respectively. Be sure to also set the `field_type` field to `SITELINK`:

5.2.0: Create operations to associate an Asset-based extension to a campaign

```
List<CampaignAssetOperation> operations = new List<CampaignAssetOperation>();

foreach (long campaignId in campaignIds)
{
  operations.Add(new CampaignAssetOperation
  {
    Create = new CampaignAsset
    {
      Asset = assetResourceName,
      FieldType = AssetFieldTypeEnum.Types.AssetFieldType.Sitelink,
```

```
      Campaign = ResourceNames.Campaign(customerId, campaignId),
    }
  });
}
```

Then upload the operations to Google Ads using a `CampaignAssetServiceClient`:

5.2.1: Send the mutate operations and report the results

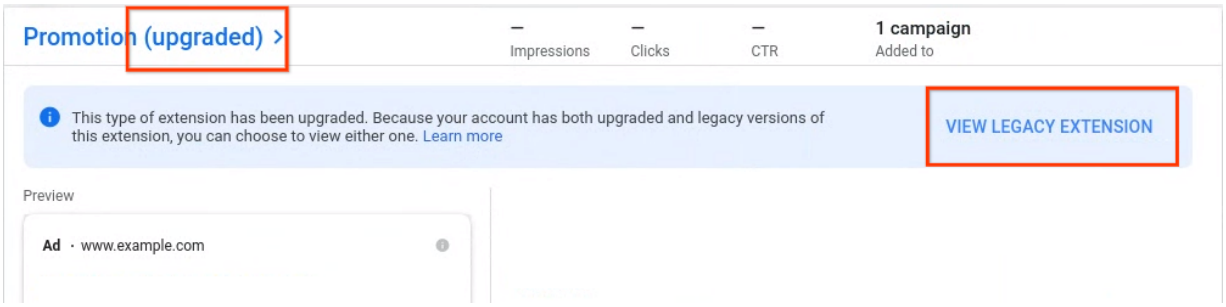
```
CampaignAssetServiceClient campaignAssetServiceClient =
  client.GetService(Services.V8.CampaignAssetService);

MutateCampaignAssetsResponse mutateCampaignAssetsResponse =
  campaignAssetServiceClient.MutateCampaignAssets(
    customerId.ToString(), operations);

foreach (MutateCampaignAssetResult result in mutateCampaignAssetsResponse.Results)
{
  Console.WriteLine($"Created CampaignAsset with resource name " +
    {result.ResourceName}");
}
```

Once you've completed this step, your new Asset-based extension is ready to serve! You can see it in Google Ads UI on the Extensions page. You may see an "Upgraded" or "Legacy" label on this page, similar to the screenshot below. Here, "Upgraded" refers to Asset-based extensions, while "Legacy" refers to Feed-based extensions.

5.2.2: UI labels for an account with Feed-based and Asset-based extensions



Note that if you have your Google Ads UI page already open, you may need to refresh it to see the new extension.

Part 6 - Remove the Feed-based extension

Now that you've created, uploaded, and associated an Asset, you'll find that your account has, essentially, two copies of the same extension. Until the sunset dates, Feed-based and Asset-based extensions can exist in your account concurrently.

You should remove the Feed-based extension. There are two key reasons to do so:

- Asset-based extensions will serve instead of Feed-based extensions of the same type. In this situation, any edits made to a Feed-based extension will not appear in your ads.
- The automatic migration process will copy all Feed-based extensions into Asset-based extensions, which will create duplicate Asset-based extensions if the original Feed-based extension is not removed.

In this part, we'll discuss the procedure for removing Feed-based extensions. We'll follow a similar procedure to the one described in the Remove an extension setting guide. We must remove both the `ExtensionSettings` and `ExtensionFeedItem` from our account.

Fortunately, we've saved the relevant information along the way!

Part 6.1: Remove the ExtensionFeedItem

We can remove the `ExtensionFeedItem` directly. Since we already have its resource name, we can simply construct an `ExtensionFeedItemOperation` and send it to the `ExtensionFeedItemService`:

6.1.0: Remove an `ExtensionFeedItem` from your account

```
ExtensionFeedItemServiceClient extensionFeedItemServiceClient =
  client.GetService(Services.V8.ExtensionFeedItemService);
ExtensionFeedItemOperation extensionFeedItemOperation =
  new ExtensionFeedItemOperation
  {
    Remove = extensionFeedResourceName
  };
MutateExtensionFeedItemsResponse response =
  extensionFeedItemServiceClient.MutateExtensionFeedItems(customerId.ToString(),
    new[] { extensionFeedItemOperation });
```

Part 6.2: Remove ExtensionSettings

We also need to remove any `ExtensionSettings` that associated the Feed-based extension with any ad groups, campaigns, or accounts. This is a bit more complicated; each ExtensionSetting can associate multiple `ExtensionFeedItems` of a specific type with a single ad group, campaign, or account. These are represented by the `ExtensionSetting`'s `extension_type` and `extension_feed_items` fields. See, for example, the `AdGroupExtensionSetting` reference:

6.2.0: Field descriptions for `AdGroupExtensionSetting`

`extension_type`	`ExtensionType` Immutable. The extension type of the ad group extension setting.

`extension_feed_items[]`	`string`
	
	The resource names of the extension feed items to serve under the ad group. ExtensionFeedItem resource names have the form:
	
	`customers/{customer_id}/extensionFeedItems/{feed_item_id}`

Therefore, there should be one `AdGroupExtensionSetting` for all Sitelink extensions for each ad group. Since we've only migrated one extension, we might not be ready to remove an entire `ExtensionSetting`; we should remove it only after all of its attached `ExtensionFeedItems` have been migrated.

Thus there are two possible operations for each `ExtensionSetting` retrieved in Part 5. The `ExtensionSetting` will either be entirely removed (if the target extension is the only one it represents) or modified to exclude the target `ExtensionFeedItem`. We'll create and send to Google Ads an `ExtensionSettingOperation` for each `ExtensionSetting`:

6.2.1: Complete flow for removing or mutating `CampaignExtensionSettings`

```
CampaignExtensionSettingServiceClient campaignExtensionSettingServiceClient =
  client.GetService(Services.V8.CampaignExtensionSettingService);

List<CampaignExtensionSettingOperation> campaignExtensionSettingOperations =
  new List<CampaignExtensionSettingOperation>();

foreach (CampaignExtensionSetting campaignExtensionSetting in
  campaignExtensionSettings)
{
  // Remove the ExtensionSetting if there is only one associated extension.
  if (campaignExtensionSetting.ExtensionFeedItems.Count == 1)
  {
    campaignExtensionSettingOperations.Add(new CampaignExtensionSettingOperation
    {
      Remove = campaignExtensionSetting.ResourceName
    });
  }
  // Otherwise, there must be other extension instances represented by this
  // ExtensionSetting; we'll change its list to exclude the Feed-based extension.
  else
  {
    campaignExtensionSetting.ExtensionFeedItems.Remove(extensionFeedResourceName);
    campaignExtensionSettingOperations.Add(new CampaignExtensionSettingOperation
    {
      Update = campaignExtensionSetting,
      UpdateMask = FieldMasks.AllSetFieldsOf(campaignExtensionSetting)
    });
```

```
  }
}

MutateCampaignExtensionSettingsResponse mutateCampaignExtensionSettingResponse =
  campaignExtensionSettingServiceClient.MutateCampaignExtensionSettings(
    customerId.ToString(), campaignExtensionSettingOperations);
```

All done! You've written a program that will copy a Feed-based extension into an Asset-based extension and remove the Feed-based extension. You might want to now adapt the program to migrate other extension types, handle multiple Feed IDs (or even all instances of an extension type), or insert your own backend interactions. Consider pausing after creating new `Assets` (Part 5) to confirm that the contents of the newly created `Assets` match the legacy extensions before deletion.

Please reach out to our support team or visit our discussion forum with any issues or concerns. Follow our blog for updates about Google Ads API.