[External] Classroom add-ons "Posts" migration guide

July 2023

Introduction

This document is a step-by-step guide for migrating a Classroom add-on implementation off of the older `courses.posts` abstraction. In the new implementation, add-on endpoints are part of the `courses.announcements`, `courses.courseWork` and `courses.courseWorkMaterials` endpoints directly.

Motivation

This change was motivated by the feedback we received from developers who were interested in knowing and accessing the underlying item the `post` was hiding. More precisely, developers were interested in subscribing to change notifications on the `announcements`, `courseWork` or `courseWorkMaterials` their add-on was attached to, such as updates to the due date.

By eliminating the `posts` abstraction, we are more concretely sharing with you the actual resource the teacher has shared with their students.

Timeline

The developer changes are minor syntactic changes that should require only a couple of days of developer time. We encourage you to make this change within the next 6 months so that we can remove the old way from the API. That said, please consult with us if 6 months is unrealistic and we'll work with you on timing.

Migration

At a high-level, there are 3 main changes to the API:

1. Query parameter `postId` has been changed to `itemId`. A new query parameter, `itemType`, has been added.
2. `courses.posts.getAddOnContext` and `courses.posts.addOnAttachments` have been moved to `Announcements`, `CourseWork` and `CourseWorkMaterials` methods.
3. `courses.posts.addOnAttachments.studentSubmissions` has been moved to `courses.courseWork.addOnAttachments.studentSubmissions`.

Query Params

For each iframe where you previously received **`postId`** as a query parameter, you will now receive an **`itemId`**. The value will be the same, only the parameter name has changed. For a period of time, you will receive both so that your add-on will not break during the migration period.

To be clear, **`itemId == postId`**.

In addition to the **`itemId`**, you will also receive an **`itemType`** parameter which will have one of 3 values: `announcements`, `courseWork`, or `courseWorkMaterials`.

Before the migration window:

```
?addOnToken={token}&courseId={courseId}&postId={postId}
```

During the migration window you will receive both **`postId`** and **`itemId`** (note: same value):

```
?addOnToken={token}&courseId={courseId}&postId={postId}&itemId={itemId}&itemType={itemType}
```

After the migration window:

```
?addOnToken={token}&courseId={courseId}&itemId={itemId}&itemType={itemType}
```

Making a helper function

You will need to know the correct parent resource for the add-on API calls you make. This is provided for you as the **`itemType`** query parameter. A reusable helper function will make this easier for you.

One possible way you could implement this:

```
# ------------------------------------------------
# Get Service
# Get the correct service based on item type
#
def get_parent_resource(item_type):
   """Returns the right service based on item type."""

   # Get the Google Classroom service
   classroom_service = ch._credential_handler.get_classroom_service()

   match item_type:
      case "announcements":
```

```
        return classroom_service.courses().announcements()
      case "courseWorkMaterials":
        return classroom_service.courses().courseWorkMaterials()
      case _:
        return classroom_service.courses().courseWork()
```

GetAddOnContext

In order to call **`getAddOnContext`**, you will need to know the correct parent resource. This is provided for you as the **`itemType`** query parameter.

One possible way you could implement this:

```
# Choose the correct service based on itemType
parent = get_parent_resource(item_type)

# Get the add-on context using the correct service
resp = parent.getAddOnContext(
      courseId=course_id,
      itemId=item_id,
      addOnToken=add_on_token
   ).execute()
```

Create add-on attachments

Similar to getAddOnContext, you will also need to use the correct parent resource.

One possible way you could implement this:

```
# Build request body
body = {
   'courseId': courseId,
   'parentId': parentId,
   'title': 'Title of attachment',
   'teacherViewUri': {'uri': 'https://example.com/teacherView?id=123'},
   'studentViewUri': {'uri': 'https://example.com/studentView?id=123'},
   'maxPoints': 100
}

# Choose the correct service based on itemType
parent = get_parent_resource(item_type)

# Create the add-on context using the correct service
resp = parent.addOnAttachments().create(
      courseId=course_id,
      parentId=parent_id,
      body=body
```

```
   ).execute()
```

Patching add-on attachments

One possible way you could implement this:

```
# Build request body and update mask
update_mask = 'AddOnAttachment.teacherViewUri'
body = {'teacherViewUri': {'uri': 'https://example.com/teacherView?id=456'}}

# Choose the correct service based on itemType
parent = get_parent_resource(item_type)

# Create the add-on context using the correct service
resp = parent.addOnAttachments().patch(
      courseId=course_id,
      parentId=parent_id,
      attachmentId=attachment_id,
      updateMask=update_mask,
      body=body
   ).execute()
```

Patching student submissions

Student submissions **only** exist on `Courses.CourseWork` resources.

One possible way you could implement this:

```
# Build request body and update mask
update_mask = 'maxPoints'
body = {
   'maxPoints': 100
}

# Only courseWork add-ons support student submissions
parent = get_parent_resource("courseWork")

# Create the add-on context using the correct service
resp = parent.addOnAttachments().studentSubmissions().patch(
      courseId=course_id,
      parentId=parent_id,
      attachmentId=attachment_id,
      updateMask=update_mask,
      body=body
   ).execute()
```