# The Definitive Guide to Programmatic Google Slides Templating: Deconstructing the files.copy Failure and Mastering the batchUpdate Workflow

## Executive Summary: The Definitive Solution

This report provides a definitive analysis and solution for the programmatic creation of Google Slides presentations from a template. The central challenge investigated is the 400 FAILED_PRECONDITION error ("This operation is not supported for this document") encountered when attempting to use the Google Drive API's files.copy method.

Direct Answer and Error Explanation:
The developer's core problem is not a limitation of the Google Slides or Drive APIs. The drive.files.copy method is, in fact, the correct and officially documented method for copying Google Slides presentations.1
The FAILED_PRECONDITION error is a misleading diagnostic. Analysis reveals this error message is a generic response that abstracts the true underlying cause: a **file permission and ownership issue**. The error indicates that the authenticated user (acting through the web application's OAuth flow) does not have sufficient permission to copy the specified templateId. This typically occurs when the user is only a reader or commenter on a template file whose owner has disabled the "Viewers and commenters can see the option to download, print, and copy" setting. The "precondition" that failed was not one of file type, but of the user's canCopy permission on that specific file.

The Best Practice Workflow:
The most robust and efficient solution involves a three-step process that combines the Drive and Slides APIs. The developer's proposed "workaround" (having the user copy the template first) is not a workaround but the required pre-condition for this workflow.

1. **Step 1: Copy (Drive API):** The application must ensure the authenticated user is the owner of the template file it intends to copy. The most reliable method is to instruct the user to "Make a Copy" of the master template to their own Google Drive and then provide the ID of *their copy* to the application. The application then successfully calls gapi.client.drive.files.copy on this user-owned file ID.
2. **Step 2: Identify (Slides API):** The application calls gapi.client.slides.presentations.get on the newly created copy. To fulfill the workflow of duplicating a specific slide (e.g., "Slide 2"), this call is used to retrieve the unique objectId of the target slide. A field mask (fields=slides.objectId) should be used to optimize this call for performance.[2]
3. **Step 3: Modify (Slides API):** The application uses gapi.client.slides.presentations.batchUpdate. This single, atomic API call is used to submit a list of requests. By leveraging the duplicateObject request's objectIds map [3], the application can duplicate the template slide multiple times while simultaneously assigning unique, predictable IDs. These new IDs are then used *in the same batch request* to target replaceAllText requests, populating the new slides with data.

Rejection of Alternatives:
This report will also demonstrate that the logical alternatives—such as reading the template's structure and "recreating" it in a blank presentation, or attempting to apply an external template/master—are technically unfeasible. The presentations.create method explicitly ignores all content, master, and layout data passed in the request body.5 Furthermore, the Slides API provides no endpoint to copy objects or masters between presentations.8 The drive.files.copy method is the only viable starting point.

---

# Part 1: Deconstructing the drive.files.copy Failure

The error 400 FAILED_PRECONDITION: This operation is not supported for this document is one of the most common and confusing blockers for developers integrating with Google Workspace. A deep analysis of this error reveals that it is a generic message abstracting a more specific, underlying cause.

## 1.1 The "Not Supported" Error: A Misleading Diagnostic

The developer's initial assumption—that drive.files.copy might not support Google Slides—is logical, given the error text. However, this assumption is incorrect. The error message itself is a "red herring."

Evidence from various developer forums and issue trackers shows this *exact* error message appearing when developers attempt to use an API on an incompatible file format.[9] For example:

- An attempt to use the Google Sheets API's spreadsheets.values.get method on a raw .xlsx file uploaded to Drive (which is not a native Google Sheet) will fail with this error.[9]
- An attempt to use the Google Docs API's documents.get method on a raw .docx file (which is not a native Google Doc) will also fail with this error.[10]

In these cases, the API is correctly stating that the operation (e.g., "get spreadsheet values") is "not supported for this document" (e.g., an Excel file). The "precondition" that failed is that the file was not in the native format the API expected.

This leads to the *first-order insight*: developers are trained to see this error and think, "I am using the wrong API or a non-native file." This is precisely what makes it a red herring in the developer's current situation. They *are* using the correct API (the Drive API for a file-level operation) on a native Google Slides file (MIME type application/vnd.google-apps.presentation). The error is not about file *type*; it is about file *state*.

## 1.2 The Root Cause: Ownership vs. Read Access in the Drive API

The true "precondition" that failed is related to the authenticated user's permissions on the templateId. The developer's OAuth scopes (https://www.googleapis.com/auth/drive and https://www.googleapis.com/auth/presentations) are correct and sufficient. These scopes grant the application *full permission* to perform any action *that the user is allowed to perform*. This is the key distinction.

The Google Drive permission model is granular, with roles such as owner, writer, commenter, and reader.[13] Crucially, the ability to *copy* a file is a discrete permission that a file's owner can control. In the Google Drive UI, this is the checkbox: "Viewers and commenters can see the option to download, print, and copy."

If a user is only a reader on a corporate template, and the owner has *disabled* this setting, the user cannot manually "Make a Copy." When the developer's application, acting on behalf of this user, attempts to call drive.files.copy, the Drive API backend performs the following check:

1. Does the user (via the app's OAuth token) have writer or owner access? If yes, allow the copy.
2. If the user is only a reader or commenter, does the file's metadata include the canCopy permission for this user?

3. In this case, the templateId refers to a file where this permission is false.

The Drive API, upon finding that the user lacks the canCopy permission, should ideally return a 403 Forbidden error with the message, "You do not have permission to copy this file." Instead, it returns the less-specific 400 FAILED_PRECONDITION. This API behavior is a known source of developer confusion.[15] The "operation" (copy) is "not supported for this document" *because the document's permissions (its state) forbid it for this user*.

## 1.3 Definitive Answer: drive.files.copy *Does* Work for Google Slides

The most critical question is whether drive.files.copy works for Slides at all. The definitive answer is **YES**.

The official Google Workspace developer documentation, in its own code samples for the Slides API, explicitly uses the Drive API's copy method as the standard, supported way to duplicate a presentation. The documentation provides clear examples in Python and Ruby [1]:

- Python example: drive_response = ( files().copy(fileId=presentation_id, body=body).execute() ) [1]
- Ruby example: drive_response = drive_service.copy_file(presentation_id, body) [1]

This evidence is conclusive. The drive.files.copy endpoint is the correct and intended method.

This directly validates the developer's "workaround" hypothesis: "If user manually copies template first... Would Drive API copy work on user's own files?" **Yes.** This is not a workaround; it is the **required pre-condition** for the application's flow.

1. When the user manually selects "File > Make a Copy" on the original template (which they can do, as the owner cannot disable this for *themselves*), they create a new presentation.
2. The user becomes the owner of this new copy.
3. When the application receives the ID of *this* new, user-owned copy, the drive.files.copy call will succeed every time, because an owner always has permission to copy their own files.

It is also important to contrast this client-side OAuth flow with a server-side service account. If this operation were run using a service account, the copied presentation would be created in the service account's own, private Google Drive, making it invisible to the user unless the service account's code *also* included logic to create permissions and share the file back to the user.[16] The developer's current client-side gapi approach is the correct architecture, as it correctly places the new copy in the authenticated user's "My Drive" folder.

---

# Part 2: The Best Practice Workflow: A Complete Implementation Guide

The following three-step process is the robust, production-ready, and API-supported "best practice" for implementing the developer's required workflow. It correctly uses the Drive API for file-level operations and the Slides API for in-presentation content manipulation.

## 2.1 Step 1: Copy the Template (The Right Way)

As established, the non-negotiable pre-condition is that the application must operate on a templateId that the authenticated user either *owns* or has explicit canCopy permissions for. The most reliable application flow is to instruct the user to "Make a Copy" of the official template, and then provide the URL/ID of *their copy* to the application.

Once this pre-condition is met, the copy operation is a standard call to the Drive API v3 files.copy method.[17]

- **API Endpoint:** gapi.client.drive.files.copy
- **Method:** POST
- **Path:** https://www.googleapis.com/drive/v3/files/{fileId}/copy
- **Request Body:** A simple File resource object [17], containing, at minimum, the name for the new presentation.

**JavaScript Code Example (Client-Side gapi):**

JavaScript

```
/**
 * Copies a Google Presentation using the Drive API.
 * CRITICAL: The authenticated user MUST be the owner of the templateId
 * or have explicit 'canCopy' permissions.
 *
 * @param {string} templateId - The ID of the presentation to copy (user-owned).
 * @param {string} title - The title for the new presentation.
```

```
 * @returns {Promise<string>} - The file ID of the newly created presentation.
 */
async function copyPresentation(templateId, title) {
  try {
    // Note: The developer's original code used gapi.client.request.
    // This example uses the loaded Drive API client, which is cleaner.
    const response = await gapi.client.drive.files.copy({
      fileId: templateId,
      resource: {
        name: title
      }
    });

    // The response body contains a File resource
    // https://developers.google.com/drive/api/reference/rest/v3/files
    const newPresentationId = response.result.id;
    console.log('Copy successful. New presentation ID:', newPresentationId);
    return newPresentationId;

  } catch (err) {
    // This is where the developer's 400 error would be caught.
    // We can now provide a more useful error message to the user.
    console.error('Error copying presentation:', err);
    if (err.status === 400 |

| err.status === 403) {
      throw new Error(
        'Failed to copy presentation. Please ensure you are the owner of the ' +
        'template file and have provided the ID of *your* copy.'
      );
    }
    throw new Error('An unknown error occurred during the copy operation.');
  }
}
```

## 2.2 Step 2: Identify the Target Slide

After Step 1, the application has a newPresentationId. The developer's workflow requires duplicating "Slide 2" multiple times. In the Slides API, all pages and page elements are referenced by a unique objectId, not by their index.[18] The application cannot assume the

objectId of "Slide 2". It must first retrieve it.

- **API Endpoint:** gapi.client.slides.presentations.get [19]
- **Method:** GET
- **Path:** https://slides.googleapis.com/v1/presentations/{presentationId}

A call to this endpoint with no parameters will return the *entire* presentation object, including all masters, layouts, and page element JSON. This is extremely wasteful if the only required information is the list of slide IDs.

To optimize this, the API request must use a *field mask* to request *only* the objectId for each slide. The slides.objectId field will return a list of slide IDs *in presentation order*.[2]

**JavaScript Code Example (Client-Side gapi):**

JavaScript

```
/**
 * Finds the unique objectId of a specific slide by its index.
 * Note: Slide 2 is at index 1.
 *
 * @param {string} presentationId - The ID of the presentation to query.
 * @param {number} slideIndex - The zero-based index of the slide (e.g., 1 for "Slide 2").
 * @returns {Promise<string>} - The unique objectId of the target slide.
 */
async function getSlideObjectId(presentationId, slideIndex) {
  try {
    const response = await gapi.client.slides.presentations.get({
      presentationId: presentationId,
      // Use a field mask for a massive performance improvement.
      // This requests *only* the objectId for each slide.
      fields: 'slides.objectId'
    });

    // The response will look like:
    // { "slides": [ { "objectId": "p1" }, { "objectId": "p2" },... ] }
    const slides = response.result.slides;

    if (!slides) {
      throw new Error('No slides found in this presentation.');
    }
```

```
    if (slides.length > slideIndex) {
      const targetSlideId = slides[slideIndex].objectId;
      console.log(`Found objectId for slide at index ${slideIndex}:`, targetSlideId);
      return targetSlideId;
    } else {
      throw new Error(`Slide index ${slideIndex} is out of bounds. Presentation only has ${slides.length} slides.`);
    }
  } catch (err) {
    console.error('Error getting slide objectId:', err);
    throw new Error('Failed to retrieve slide information from the new presentation.');
  }
}
```

## 2.3 Step 3: Duplicate and Modify Slides (The batchUpdate)

This is the core of the programmatic workflow. The Slides API is designed around the presentations.batchUpdate method, which allows an application to send a list of individual requests (e.g., create slide, delete text, duplicate object) to be processed as a single, atomic operation.[18] If any request in the batch fails, the entire update is rolled back.

To implement the developer's workflow, the batch will contain two types of requests:

1. **duplicateObject:** This request is used to copy a slide or page element.[3]
2. **replaceAllText:** This request finds and replaces text, and can be scoped to specific pages.[23]

A naive implementation would require multiple API calls: one batchUpdate to duplicate the slide, a get call to find the new slide's ID, and then another batchUpdate to add the text.

The **professional technique** avoids this. The DuplicateObjectRequest JSON includes an optional objectIds map.[3] This map allows the developer to *pre-define* the objectId of the new slide. For example: {"original_slide_id": "my_new_unique_slide_id"}.

This is a critically important feature. It allows the application to *chain* requests in the *same batch*. The application can submit a duplicateObject request that defines a new ID, and then, in the same list, submit a replaceAllText request that *targets that new ID*. The API processes the requests in order, so the new slide exists by the time the text replacement request is processed.[20]

This technique reduces what could be 2N + 1 API calls (where N is the number of duplicates) into a **single, atomic API call**.

**JavaScript Code Example (Full Workflow):**

JavaScript

```
/**
 * Implements the full "Copy, Duplicate, Modify" workflow.
 *
 * @param {string} userOwnedTemplateId - The ID of the template the user OWNS.
 * @param {string} newPresentationTitle - The title for the new deck.
 * @param {number} slideIndexToDuplicate - The index of the slide to use as a template (e.g., 1 for "Slide 2").
 * @param {Array<object>} slideData - An array of data objects. A slide will be created for each item.
 *                                    Example:
 * @param {string} titlePlaceholder - The placeholder text in the template (e.g., "{{TITLE}}").
 * @param {string} bodyPlaceholder - The placeholder text in the template (e.g., "{{BODY}}").
 */
async function createPresentationFromTemplate(
  userOwnedTemplateId,
  newPresentationTitle,
  slideIndexToDuplicate,
  slideData,
  titlePlaceholder = '{{CONTENT_TITLE}}',
  bodyPlaceholder = '{{CONTENT_BODY}}'
) {
  try {
    // Step 1: Copy the presentation (using function from 2.1)
    console.log('Step 1: Copying presentation...');
    const newPresentationId = await copyPresentation(userOwnedTemplateId, newPresentationTitle);

    // Step 2: Get the objectId of the template slide (using function from 2.2)
    console.log(`Step 2: Finding template slide at index ${slideIndexToDuplicate}...`);
    const templateSlideId = await getSlideObjectId(newPresentationId, slideIndexToDuplicate);

    // Step 3: Prepare the batchUpdate requests
    console.log('Step 3: Preparing batch update requests...');
    const requests =;

    for (let i = 0; i < slideData.length; i++) {
```

```
    const data = slideData[i];
    // Generate a unique, predictable ID for the new slide
    const newSlideId = `duplicated_slide_${i}_${Date.now()}`;

    // Request A: Duplicate the template slide
    requests.push({
      duplicateObject: {
        objectId: templateSlideId,
        // This objectIds map is the key technique.
        // It maps the original slide ID to the new ID we want to create.
        objectIds: {
         : newSlideId
        }
      }
    });

    // Request B: Replace the title placeholder on the *new* slide
    requests.push({
      replaceAllText: {
        containsText: { text: titlePlaceholder, matchCase: true },
        replaceText: data.title,
        // Scope this replacement to *only* the slide we just created
        pageObjectIds:
      }
    });

    // Request C: Replace the body placeholder on the *new* slide
    requests.push({
      replaceAllText: {
        containsText: { text: bodyPlaceholder, matchCase: true },
        replaceText: data.body,
        pageObjectIds:
      }
    });
  }

  // Optional: After all duplicates are made, delete the original template slide
  requests.push({
    deleteObject: {
      objectId: templateSlideId
    }
  });
```

```
    // Step 4: Execute the single, atomic batch update
    if (requests.length > 0) {
      console.log(`Step 4: Executing batch update with ${requests.length} requests...`);
      await gapi.client.slides.presentations.batchUpdate({
        presentationId: newPresentationId,
        resource: {
          requests: requests
        }
      });
    }

    const presentationUrl = `https://docs.google.com/presentation/d/${newPresentationId}/`;
    console.log(`Workflow complete! New presentation available at: ${presentationUrl}`);
    return presentationUrl;

  } catch (err) {
    console.error('Full templating workflow failed:', err.message);
  }
}
```

---

# Part 3: Analysis of API-Based Alternatives (And Their Limitations)

To provide an exhaustive report, it is necessary to analyze the logical alternatives proposed by the developer. These lines of inquiry are common, and understanding their limitations solidifies *why* the workflow in Part 2 is the only viable best practice.

## 3.1 Alternative 1: The "Read and Recreate" Strategy (Unfeasible)

The developer's first alternative is logical: "Can I use Slides API to GET the template presentation structure... and apply those to a new blank presentation?"

**Reading the Template:** This part is technically possible. A call to presentations.get (without a field mask) returns a complete Presentation JSON object.[2] This object contains comprehensive details about the presentation, including [7]:

- masters: An array of all slide masters, defining common page elements and properties.[18]
- layouts: An array of all layouts, which inherit from masters.[25]
- pageSize: The dimensions of the presentation.
- slides: An array of all slides, containing their pageElements (shapes, text, etc.).[25]
- Theme and color information, which can be extracted from the masters.[28]

**The Critical Flaw (Creating):** The entire strategy fails at the next step. The presentations.create method is deceptively simple and does not function as a "re-creation" endpoint.

The official documentation for presentations.create is explicit and unambiguous:

> "Creates a blank presentation using the title given in the request. If a presentationId is provided, it is used as the ID of the new presentation. Otherwise, a new ID is generated. **Other fields in the request, including any provided content, are ignored.**" [7]

This is confirmed by numerous developer reports and an analysis of the API.[5] An application *cannot* pass the masters, layouts, or slides arrays obtained from a presentations.get call into a presentations.create call. The API will silently ignore all of it and return a blank, default-themed presentation.[1]

This leaves only one "re-creation" path:

1. Call presentations.create to create a *blank* presentation.
2. Call presentations.get on the *template* to get its full JSON.
3. The developer would then have to write a complex translation layer to parse the *entire* GET JSON and convert every single master, layout, placeholder, shape, text run, and theme color into a massive, corresponding list of presentations.batchUpdate requests (e.g., createSlide, createShape, updatePageProperties, updateTextStyle).[25]

This "re-creation" path is a notorious developer trap. As noted by developers who have attempted this, the GET schemas are not 1:1 with the batchUpdate schemas, and the process is brittle, manual, and fails to capture high-fidelity details.[25] It is not a viable or scalable solution.

## 3.2 Alternative 2: "Import/Apply Template" After Creation (Not Supported)

The developer's second alternative is also logical: "Is there an applyTemplate or similar

method? Can I reference master slides from another presentation?"

**Definitive Answer:** No. The Google Slides v1 REST API has no endpoint or method for applyTemplate, importTheme, or copyMaster. An review of the API's REST resource documentation confirms these methods do *not* exist.[26] This functionality, which *does* exist in the Google Slides UI ("Import slides") and the older Google Apps Script environment (SlidesApp.openById(...) [35]), has not been implemented in the modern v1 REST API.

A developer might see the CreateSlideRequest (used in a batchUpdate) and believe it offers a solution.[36] This request object contains a slideLayoutReference.[37] However, this slideLayoutReference union can only contain one of two things:

1. predefinedLayout: A standard, built-in layout (e.g., TITLE_AND_BODY, BLANK).
2. layoutId: The objectId of a layout *that already exists within the presentation's own layouts array*.

It is impossible to use createSlide to reference a layoutId from an *external* presentationId. The API is explicit: "Copying objects across presentations isn't supported just yet".[8] Any attempted workaround, such as manually reading the elements from a template slide and creating them on a new slide [39], is simply a less-reliable version of the "Read and Recreate" strategy.

## 3.3 Table: Comparison of Presentation Creation Strategies

The following table summarizes the analysis, providing an at-a-glance justification for the recommended "Best Practice" workflow.

| Strategy | Core API Call(s) | Viability | Complexity | Key Limitation |
|---|---|---|---|---|
| **1. Drive Copy (Best Practice)** | gapi.client.drive.files.copy | **High** | **Low** | Requires the authenticated user to have owner or copy permissions on the source file. |
| **2. Read & Recreate** | presentations.get &arr; | **Extremely Low** | **Extreme** | presentations.create ignores |

| | | | | |
|---|---|---|---|---|
| | presentations.create &arr; presentations.batchUpdate | | | all content/masters/layouts passed to it.[5] Requires a massive, brittle translation layer to convert GET JSON into batchUpdate requests.[25] |
| **3. Apply External Template** | presentations.create &arr; ??? | **Not Possible** | **N/A** | No applyTemplate, importTheme, or importMaster endpoint exists in the Slides v1 REST API.[26] createSlide cannot reference layout IDs from external presentations.[8] |

---

# Part 4: Conclusion and Final Recommendations

The analysis confirms a clear and definitive path forward for developers seeking to programmatically generate Google Slides presentations from a template.

Summary of Findings:
The 400 FAILED_PRECONDITION error encountered during the initial drive.files.copy attempt is not an API limitation or bug. It is a misleading error message that abstracts the true cause: the authenticated user lacks the necessary canCopy permission on the target templateId. The drive.files.copy method is, in fact, the correct and only supported method for creating a high-fidelity copy of a Google Slides presentation.1

The Recommended Path:
The most robust, efficient, and API-supported solution is the three-step "Copy, Identify, Modify" workflow detailed in Part 2 of this report:

1. **Copy:** Use drive.files.copy on a template ID that the user *owns*. The application's UI/UX must guide the user to provide a file for which this pre-condition is met.
2. **Identify:** Use slides.presentations.get with a fields=slides.objectId field mask to efficiently find the objectId of the template slide (e.g., "Slide 2") within the new copy.[2]
3. **Modify:** Use a *single* slides.presentations.batchUpdate call to perform all necessary slide duplications and text replacements.[20]

The "One Call" Advantage:
The key to a professional-grade implementation lies in Step 3. By leveraging the objectIds map within the DuplicateObjectRequest 3, the application can pre-define the objectId of each new slide. This allows for subsequent requests in the same batch, such as replaceAllText 24, to target these new slides immediately. This "chaining" technique transforms a potentially slow, high-traffic workflow into a single, atomic, and high-performance API call.
Final Warning:
Developers must avoid the "Read and Recreate" and "Import Template" alternatives. As demonstrated in Part 3, these logical paths are well-documented technical dead ends. The presentations.create method is explicitly documented to ignore all content, master, and layout data 5, and the API provides no mechanism for copying masters or layouts between presentations.8 The drive.files.copy method is, and will remain, the correct foundation for any Google Slides templating workflow.

## Works cited


1. Create and manage presentations | Google Slides, accessed November 15, 2025, https://developers.google.com/workspace/slides/api/guides/presentations
2. Basic reading | Google Slides, accessed November 15, 2025, https://developers.google.com/workspace/slides/api/samples/reading
3. Slide operations - Google for Developers, accessed November 15, 2025, https://developers.google.com/workspace/slides/api/samples/slides
4. Google Slide API-How do i duplicate a slide several times and creating a unique object id each time - Stack Overflow, accessed November 15, 2025, https://stackoverflow.com/questions/59284060/google-slide-api-how-do-i-duplicate-a-slide-several-times-and-creating-a-unique
5. How To Define Masters In A New Presentation With Google Slides API? - Stack Overflow, accessed November 15, 2025, https://stackoverflow.com/questions/54270655/how-to-define-masters-in-a-new-presentation-with-google-slides-api
6. How to apply custom layout in Google Slides API when I create a presentation, accessed November 15, 2025, https://stackoverflow.com/questions/58824017/how-to-apply-custom-layout-in-google-slides-api-when-i-create-a-presentation

7. Method: presentations.create | Google Slides, accessed November 15, 2025, https://developers.google.com/workspace/slides/api/reference/rest/v1/presentations/create
8. Google Slide API - copy an object from a file to another file - Stack Overflow, accessed November 15, 2025, https://stackoverflow.com/questions/48273824/google-slide-api-copy-an-object-from-a-file-to-another-file
9. ERROR: "failedPrecondition: this operation is not supported for this document", accessed November 15, 2025, https://docs.holistics.io/faqs/error-when-working-with-google-spreadsheet
10. Failed to get docx content using documents.get(fileID) with error Error 400: This operation is not supported for this document, failedPrecondition [242205302] - Google Issue Tracker, accessed November 15, 2025, https://issuetracker.google.com/issues/242205302
11. Read operation in google sheets error - Questions - n8n Community, accessed November 15, 2025, https://community.n8n.io/t/read-operation-in-google-sheets-error/12670
12. This operation is not supported for this document - Sheets API - Stack Overflow, accessed November 15, 2025, https://stackoverflow.com/questions/64668337/this-operation-is-not-supported-for-this-document-sheets-api
13. Google drive permission issue while using Javascript API - Stack Overflow, accessed November 15, 2025, https://stackoverflow.com/questions/29621429/google-drive-permission-issue-while-using-javascript-api
14. Google Drive API Permissions error when attempting to create permissions - Stack Overflow, accessed November 15, 2025, https://stackoverflow.com/questions/73088610/google-drive-api-permissions-error-when-attempting-to-create-permissions
15. drive.files.copy failed with error - Google Groups, accessed November 15, 2025, https://groups.google.com/g/google-apps-script-community/c/UjNvjioZgOo
16. unable to copy a google slide file using google drive api - Stack Overflow, accessed November 15, 2025, https://stackoverflow.com/questions/64626446/unable-to-copy-a-google-slide-file-using-google-drive-api
17. Method: files.copy | Google Drive, accessed November 15, 2025, https://developers.google.com/workspace/drive/api/reference/rest/v3/files/copy
18. Introduction | Google Slides, accessed November 15, 2025, https://developers.google.com/workspace/slides/api/guides/overview
19. Method: presentations.get | Google Slides, accessed November 15, 2025, https://developers.google.com/workspace/slides/api/reference/rest/v1/presentations/get
20. Batch requests | Google Slides, accessed November 15, 2025, https://developers.google.com/workspace/slides/api/guides/batch
21. DuplicateObjectRequest (Google Slides API v1-rev20250401-2.0.0), accessed

November 15, 2025, https://googleapis.dev/java/google-api-services-slides/latest/com/google/api/services/slides/v1/model/DuplicateObjectRequest.html
22. DuplicateObjectRequest (Google Slides API v1 (Rev. 399) 1.25.0), accessed November 15, 2025, https://developers.google.com/resources/api-libraries/documentation/slides/v1/java/latest/com/google/api/services/slides/v1/model/DuplicateObjectRequest.html
23. Text from Sheets ⟶ Duplicate Google Slides slide ⟶ Paste text into new slide - Hire a Pro, accessed November 15, 2025, https://community.make.com/t/text-from-sheets-duplicate-google-slides-slide-paste-text-into-new-slide/93787
24. Update text of an object in google slides using google slides API - Stack Overflow, accessed November 15, 2025, https://stackoverflow.com/questions/46578375/update-text-of-an-object-in-google-slides-using-google-slides-api
25. Experiments with the Google Slides API to recreate slides - Benjamin Tumbleson, accessed November 15, 2025, https://www.bentumbleson.com/experiments-with-the-google-slides-api-to-recreate-slides/
26. REST Resource: presentations | Google Slides, accessed November 15, 2025, https://developers.google.com/workspace/slides/api/reference/rest/v1/presentations
27. Google Slides API cannot read content of slide from an instance of page (Node.Js), accessed November 15, 2025, https://stackoverflow.com/questions/53860533/google-slides-api-cannot-read-content-of-slide-from-an-instance-of-page-node-js
28. Slides Service | Apps Script - Google for Developers, accessed November 15, 2025, https://developers.google.com/apps-script/reference/slides
29. Element operations | Google Slides, accessed November 15, 2025, https://developers.google.com/workspace/slides/api/samples/elements
30. How to change a Google Slides presentation theme via Slides API? - Stack Overflow, accessed November 15, 2025, https://stackoverflow.com/questions/66139690/how-to-change-a-google-slides-presentation-theme-via-slides-api
31. How to Use the Google Slides API to Create Presentations Automatically - FlashDocs API, accessed November 15, 2025, https://www.flashdocs.com/post/how-to-use-the-google-slides-api-to-create-presentations-automatically
32. Create Presentation Slides with AI | by Ivan Campos | Sopmac AI - Medium, accessed November 15, 2025, https://medium.com/sopmac-ai/create-presentation-slides-with-ai-eaa81cd21028
33. Google Slides API - Google for Developers, accessed November 15, 2025, https://developers.google.com/workspace/slides/api/reference/rest
34. REST Resource: presentations.pages | Google Slides, accessed November 15,

2025, https://developers.google.com/workspace/slides/api/reference/rest/v1/presentations.pages
35. Google Slides API - How way change theme via API - Stack Overflow, accessed November 15, 2025, https://stackoverflow.com/questions/52870270/google-slides-api-how-way-change-theme-via-api
36. Create a Slide - Google for Developers, accessed November 15, 2025, https://developers.google.com/workspace/slides/api/guides/create-slide
37. Requests | Google Slides, accessed November 15, 2025, https://developers.google.com/workspace/slides/api/reference/rest/v1/presentations/request
38. googleapis documentation, accessed November 15, 2025, https://googleapis.dev/nodejs/googleapis/latest/slides/interfaces/Params$Resource$Presentations$Batchupdate.html
39. How to add slides from another presentation using the Google Slides API in PHP?, accessed November 15, 2025, https://stackoverflow.com/questions/79228796/how-to-add-slides-from-another-presentation-using-the-google-slides-api-in-php