Synchronize resources efficiently

This guide describes how to implement "incremental synchronization" of calendar data. Using this method, you can keep data for all calendar collections in sync while saving bandwidth.

Overview

Incremental synchronization consists of two stages:

1. Initial full sync is performed once at the very beginning in order to fully synchronize the client's state with the server's state. The client will obtain a sync token that it needs to persist.
2. Incremental sync is performed repeatedly and updates the client with all the changes that happened ever since the previous sync. Each time, the client provides the previous sync token it obtained from the server and stores the new sync token from the response.

A **sync token** is a piece of data exchanged between the server and the client, and has a critical role in the synchronization process. An example would look like the following:

```
"nextSyncToken": "CPDAlvWDx70CEPDAlvWDx70CGAU=",
```

Initial full sync

The initial full sync is the original request for all the resources of the collection you want to synchronize. You can optionally restrict the list request using request parameters if you only want to synchronize a specific subset of resources.

In the response to the list operation, you will find a field called `nextSyncToken` representing a sync token. You'll need to store the value of `nextSyncToken`. If the result set is too large and the response gets paginated (/workspace/calendar/v3/pagination), then the `nextSyncToken` field is present only on the very last page.

You don't need to worry about any new entries appearing while you are paginating — they won't be missed. The information needed for the server to generate a correct sync token is encoded in the page token.

Incremental sync

Incremental sync allows you to retrieve all the resources that have been modified since the last sync request. To do this, you need to perform a list request with your most recent sync token specified in the **syncToken** field. Keep in mind that the result will always contain deleted entries, so that the clients get the chance to remove them from storage.

In cases where a large number of resources have changed since the last incremental sync request, you may find a **pageToken** instead of a **syncToken** in the list result. In these cases you'll need to perform the exact same list query as was used for retrieval of the first page in the incremental sync (with the exact same **syncToken**), append the **pageToken** to it and paginate through all the following requests until you find another **syncToken** on the last page. Make sure to store this **syncToken** for the next sync request in the future.

Here are example queries for a case requiring incremental paginated sync:

Original query

```
GET /calendars/primary/events?maxResults=10&singleEvents=true&syncToken=CPDAlvWD

// Result contains the following

"nextPageToken":"CiAKGjBpNDd2Nmp2Zml2cXRwYjBpOXA",
```

Retrieving next page

```
GET /calendars/primary/events?maxResults=10&singleEvents=true&syncToken=CPDAlvWD
```

The set of query parameters that can be used on incremental syncs is restricted. Each list request should use the same set of query parameters, including the initial request. For the individual restrictions on each collection, see the corresponding documentation for list requests. The response code for list queries containing disallowed restrictions is **400**.

Full sync required by server

Sometimes sync tokens are invalidated by the server, for various reasons including token expiration or changes in related ACLs. In such cases, the server will respond to an incremental request with a response code **`410`**. This should trigger a full wipe of the client's store and a new full sync.

Sample code

The snippet of sample code below demonstrates how to use sync tokens with the Java client library (/api-client-library/java/apis/calendar/v3). The first time the run method is called it will perform a full sync and store the sync token. On each subsequent execution it will load the saved sync token and perform an incremental sync.

```
private static void run() throws IOException {
  // Construct the {@link Calendar.Events.List} request, but don't execute it
  Calendar.Events.List request = client.events().list("primary");

  // Load the sync token stored from the last execution, if any.
  String syncToken = syncSettingsDataStore.get(SYNC_TOKEN_KEY);
  if (syncToken == null) {
    System.out.println("Performing full sync.");

    // Set the filters you want to use during the full sync. Sync tokens aren'
    // most filters, but you may want to limit your full sync to only a certai
    // In this example we are only syncing events up to a year old.
    Date oneYearAgo = Utils.getRelativeDate(java.util.Calendar.YEAR, -1);
    request.setTimeMin(new DateTime(oneYearAgo, TimeZone.getTimeZone("UTC")));
  } else {
    System.out.println("Performing incremental sync.");
    request.setSyncToken(syncToken);
  }

  // Retrieve the events, one page at a time.
  String pageToken = null;
  Events events = null;
  do {
    request.setPageToken(pageToken);
```

```
      try {
        events = request.execute();
      } catch (GoogleJsonResponseException e) {
        if (e.getStatusCode() == 410) {
          // A 410 status code, "Gone", indicates that the sync token is invalid
          System.out.println("Invalid sync token, clearing event store and re-sy
          syncSettingsDataStore.delete(SYNC_TOKEN_KEY);
          eventDataStore.clear();
          run();
        } else {
          throw e;
        }
      }

      List<Event> items = events.getItems();
      if (items.size() == 0) {
        System.out.println("No new events to sync.");
      } else {
        for (Event event : items) {
          syncEvent(event);
        }
      }

      pageToken = events.getNextPageToken();
    } while (pageToken != null);

    // Store the sync token from the last request to be used during the next exe
    syncSettingsDataStore.set(SYNC_TOKEN_KEY, events.getNextSyncToken());

    System.out.println("Sync complete.");
  }
```
b/calendar/sync/src/main/java/com/google/api/services/samples/calendar/sync/SyncTokenSample.java)

Legacy synchronization

For event collections, it is still possible to do synchronization in the legacy manner by preserving the value of the updated field from an events list request and then using the **`modifiedSince`** field to retrieve updated events. This approach is no longer recommended as it is more error-prone with respect to missed updates (for example if does not enforce query restrictions). Furthermore, it is available only for events.

Last updated 2025-07-30 UTC.