Cloud Save

The Cloud Save service allows you to store application data for each user of an application on Google's servers. Your application can retrieve and update this user data from Android devices, iOS devices, or web applications by using the Cloud Save APIs.

This service makes it possible to synchronize data for each user of an application across multiple devices and platforms. For example, if you have a game that runs on Android, iOS, and the web, you can use the Cloud Save service to allow a user to start a game on their Android phone, and then continue playing on a tablet, iOS device, or web browser without losing any of their progress. This service can also be used to ensure that a user's game play continues from where it left off even if their device is lost, destroyed, or traded in for a newer model.

Cloud Save and Google Drive

Google provides two primary developer solutions for storing user application data in the cloud: Cloud Save and the Google Drive SDK.

Cloud Save is a lightweight service designed for a specific purpose: Saving and loading of a small amount of application data for each user of the application, such as the current score and level progress in a game.

By contrast, the Google Drive SDK is designed to meet a large range of developer needs, including the ability to share files with other users, manage a set of multiple files and subdirectories, add and reply to comments, review revision history, and more.

You should choose the service best suited to your application and its needs. Here is a comparison table to help you decide:

Service	Cloud Save	Google Drive SDK
Actions	Save and load	Save, load, patch, update, delete, copy, see revision history, change permissions, share with other users, manage comments and replies, and more
Data Format	byte arrays	Custom text file formats, binary file formats and folder structure
Data Size and Quota	4 x 128KB slots (maximum of 512KB), no impact on the user's Google Drive quota	Any size up to the user's Google Drive quota, with a maximum of 10GB for a single file
Visibility	Not visible to the user, except through your application	May be visible (and editable) by the user through the Google Drive website, or the Google Drive mobile applications, depending on your implementation.
Additional libraries required?	No	• Google Drive client library for Android devices • Google APIs Client Library for Objective-C for iOS devices

Use cases	• Saving and loading application state for a user across multiple devices, such as current game score and unlocked levels	• Saving and loading complex application states larger than 512KB, or saving application states as multiple files • Uploading screenshots to Google Drive for users to view, edit, or share • Collaborative editing of files with other users

For more information on integrating Google Drive features into your application, see the Google Drive SDK documentation.

Data format

The Cloud Save service uses byte arrays as a data storage format. The data is sent and received from Google's servers as web-safe, Base64-encoded strings. On Android devices, the Base64 strings are decoded into `byte` arrays. On iOS devices, the Base64 strings are read in as `NSData` objects.

How you represent this data within your application is completely up to you, but we recommend choosing a platform-independent format so that the data can easily be read and written from any of the platforms your application supports. For instance, consider using a platform-neutral format such as a JSON-encoded string, instead of a platform-dependent mechanism such as iOS's `NSKeyedArchiver` class.

Data slots

Applications that use the Cloud Save service can use up to four different sets of data, or *slots*, to save and load data for each user. Each slot is identified by an integer, or *key*, from 0 to 3 and can hold up to 128KB of data, for a total of 512KB.

Each data slot can be updated independently of the other slots, which enables several options for managing application data. For example, you can use the slots to manage multiple saved games for a user. You could also use the slots to create separately managed data segments that can be updated independently, and thereby minimize data upload sizes and reduce network bandwidth consumption. If your application is an adventure game, for example, you can store a player's inventory in one slot, while storing the appearance of their in-game character in another. If the player then changed their character's hair color, you can then update their character's appearance without having to re-upload the player's entire inventory.

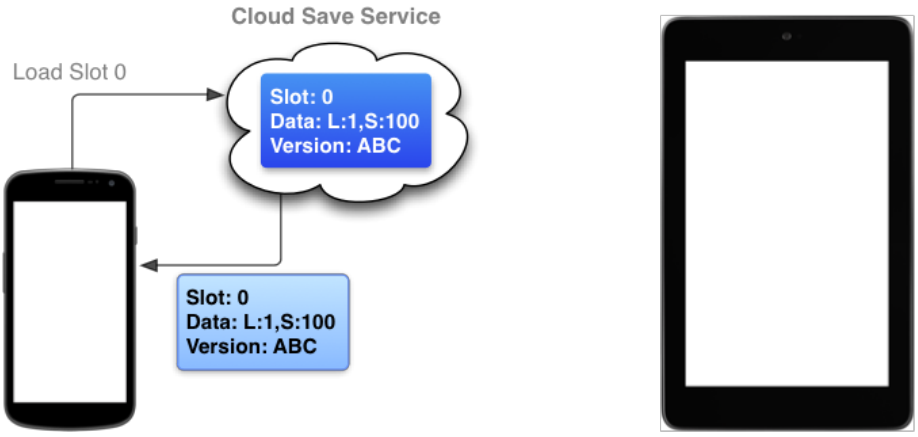
> **Note:** Data in a slot must be written as a single block. It is not possible to add or update a portion of the data in a slot. Your application must write the entire set of data in a slot each time you update it.

If you attempt to load a slot from a data slot in which your application has never saved any data using the web API, the service returns a 404 error. You can use this behavior to distinguish between a previously used slot that contains no data and a slot that has never been used. The mobile client libraries catch these errors and report them as status flags in their callback method.

Overall flow

When your application retrieves Cloud Save data from Google's servers, it receives the following information:

1. State data
2. Version identifier string for the saved state
3. Data slot identifier you requested

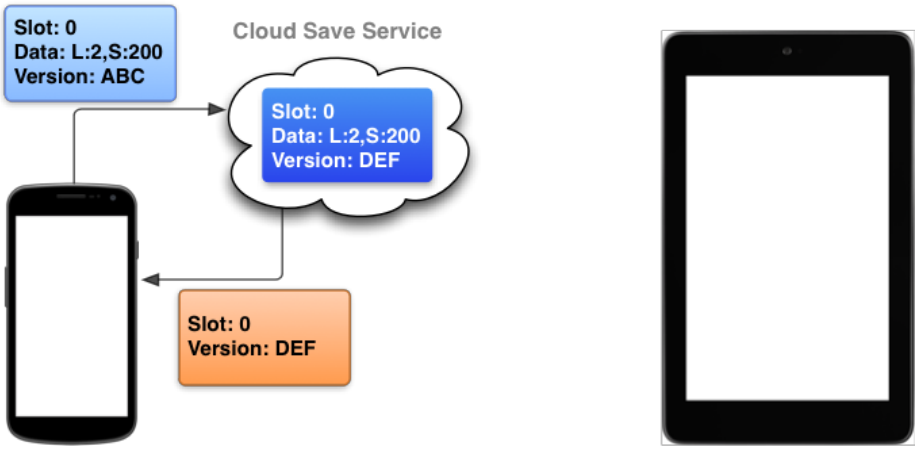


When updating your application's state data on Google's servers, you must include the following information:

1. Data slot identifier or key to specify the slot you want to update
2. Current state data you want to save
3. Last known version identifier string for the specified data slot

> **Note:** The version identifier you provide must be the same string that was returned with the last successful save to the cloud for the specified data slot.

The Cloud Save service receives this data, confirms that the version string is the same as the one currently on its servers, records the new data, and updates the version string. If the version strings are different, this indicates that a conflict has occurred, and your application must handle it. For more information on data conflicts and how to resolve them, see Conflicts.



If you are using the iOS or the Android client libraries, they maintain version strings for you. If you are accessing the Cloud Save API directly via the REST APIs, you must record and send a version string to the service whenever you attempt to update a data slot.

Conflicts

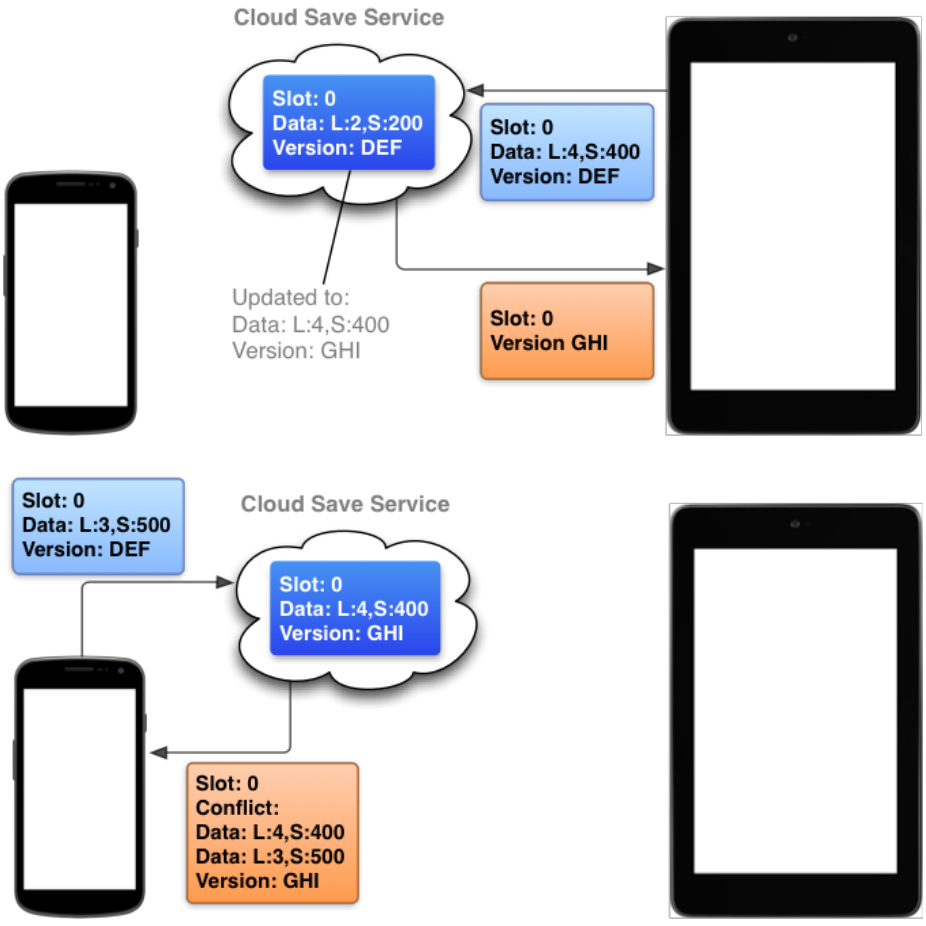
When using the Cloud Save service, your application may encounter conflicts when attempting to save data. These conflicts can occur when a user is running more than one instance of your application on different devices or computers. Your application must be able to resolve these conflicts in a way that provides the best user experience.

Typically, data conflicts occur when an instance of your application is unable to reach the Cloud Save service while attempting to load data or save it. In general, the best way to avoid data conflicts is to always load the latest data from the service when your application starts up or resumes, and save data to the service with reasonable frequency. However, it is not always possible to avoid data conflicts. Your application should make every effort to handle conflicts such that your users' data is preserved and that they have a good experience.

The following scenario describes how Cloud Save service detects and responds to data state conflict for a game application that tracks level completion and score.

1. A user runs a game on her phone and gets to level 2 with a score of 200, which the game sends to the Cloud Save service and is marked as version `DEF`.
2. The user later runs the same game on her tablet device, which retrieves the `DEF` state and updates the game progress.
3. The user continues playing the game from where she left off and achieves level 4 and a score of 400, which the game saves to the cloud and is marked as version `GHI`.
4. Later on, the user is commuting home on the train and pulls out her phone to continue playing the game. However, the train doesn't have wireless service, so the application cannot retrieve the currently saved state from the cloud.
5. The user plays the game and gets to level 3 with a score of 500.
6. Eventually, the user reaches a location where she can get wireless service and the game attempts to save her progress, sending state data and the last version string it knows about (`DEF`).
7. The Cloud Save service checks the provided version string `DEF` against the currently saved version string `GHI`, recognizes the conflict and returns a conflict message to the application.

The following illustrations summarize the update to the Cloud Save service from the table device and the resulting conflict encountered by the phone device:



At this point, the game on the user's phone has two conflicting sets of data and the application must decide how to resolve the conflict.

This scenario is the most common data conflict that your application must resolve. Other sources of conflicts may be an unexpected shutdown of your application, or attempted saves to the Cloud Save service that fail due to network problems or authentication errors. Be sure to think through other potential sources of data conflicts

for your application and provide a graceful recovery path for your users.

Resolving conflicts

As an application developer using the Cloud Save service, it is up to you to decide how best to resolve data conflicts. While the exact implementation depends on the target platform, the overall strategy for resolving conflicts is similar on all platforms.

The Cloud Save service indicates there is a conflict when you attempt to save application data to a slot with a version string that does not match the currently saved version string on Google's servers. If you are using the Android and iOS client libraries, the Cloud Save service uses a callback method in your application to signal a data conflict. The callback provides both sets of data: the local data that you are trying to save, and the data on the server that you are trying to replace. If you are using the web REST API, can retrieve the version of the data currently saved on Google's servers using a separate API call. In either case, you must resolve the data conflict by providing a set of data and the version string most recently retrieved from the Cloud Save service.

The data you send to the Cloud Save service to resolve the data conflict can be the version of the data stored in the cloud, the local set of data you currently have, or a new set of data that intelligently merges the two existing sets of data. It is important to consider how to best serve your users' needs when resolving a data conflict. Consider the data conflict received by the game application in the scenario describe in the previous section. The application has two sets of conflicting data:

1. Local progress of the game at level 3 and score of 500
2. Remote saved progress with level 4 and a score of 400

A simple solution might be to throw away the local game state because the saved version has a higher level unlocked (4). However, the remotely saved version also has a lower score than the current game state, and suddenly switching the user to a new game state might be disturbing to the user. A better solution might be to quietly unlock level 4 on the user's local device, but otherwise leave the player's game state unchanged. Make sure you think through the possibilities for your own application's data and choose an approach that provides the best experience for your users.

However you decide to resolve a data conflict, your application must call the Cloud Save API with a data set that resolves the conflict and the last version code received from the Cloud Save service.

For more information about how to implement data conflict resolution code for specific platforms, see the Android, iOS, or Web developer guides.

Security

The data maintained by your application in the Cloud Save service is protected by the same level of data security as our other cloud-based products. If you store sensitive application state data on users' local systems, you should also ensure that your application follows security best practices for storing data on those platforms.

Client-generated data can be vulnerable to malicious tampering, so your application should perform validation of data received from the service to ensure its completeness and integrity. As an additional level of protection, you may choose to encrypt sensitive application data stored with the Cloud Save service using a key that is not directly accessible to your application. On the Android platform, for example, you can place a key in a KeyStore and protect it with a password that is not stored on the device.

Last updated May 15, 2013.