Search Medium

Sign in with Google

Use your Google Account to sign in to Medium

No more passwords to remember. Signing in is fast, simple and secure.

Continue

lished in Android Developers

milie Roberts
ug 23, 2018 · 7 min read · Listen

ementing linkedPurchaseToken cor event duplicate subscriptions

use Google Play subscriptions? Make sure your back-end server ents them correctly.

scription REST APIs are the source of truth for managing user ptions. The Purchases.subscriptions API response contains an nt field called **linkedPurchaseToken.** Proper treatment of this field is for ensuring the correct users have access to your content.

oes it work?

ned in the subscriptions documentation, every new Google Play

oogl
Canadian, vegan, roller derby athlete.

Follow

More from Medium

The PyCoach in Artificial Corner
You're Using ChatGPT Wrong! Here's How to Be Ahead of 99% of ChatGPT Users

Alexander Ng... in Level Up Co...
Why I Keep Failing Candidates During Google Interviews...

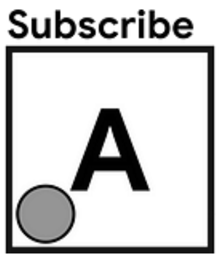
Josep Ferrer in Geek Culture
Stop doing this on ChatGPT and get ahead of the 99% of its users

Jacob Benne... in Level Up Codi...
Use Git like a senior engineer

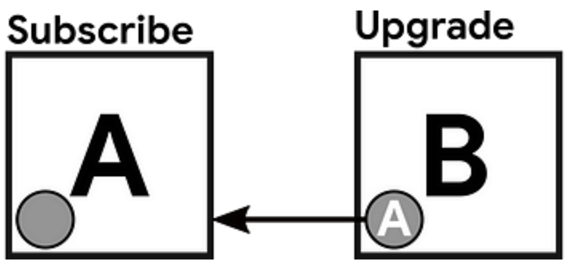
;e flow–initial purchase, upgrade, downgrade, and resignup[1]–
es a new purchase token. The **linkedPurchaseToken** field makes it
: to recognize when multiple purchase tokens belong to the same
otion.

, ***March 2021.*** *Note: the "resignup" action is no longer a concern with the*
:tion of the Resubscribe feature in Google Play Billing, available to all
ıkedPurchaseToken is still important for "upgrade" and "downgrade"

nple, a user buys a subscription and receives a purchase token A.
:edPurchaseToken field (grey circle) will not be set in the API
e because the purchase token belongs to a brand new subscription.



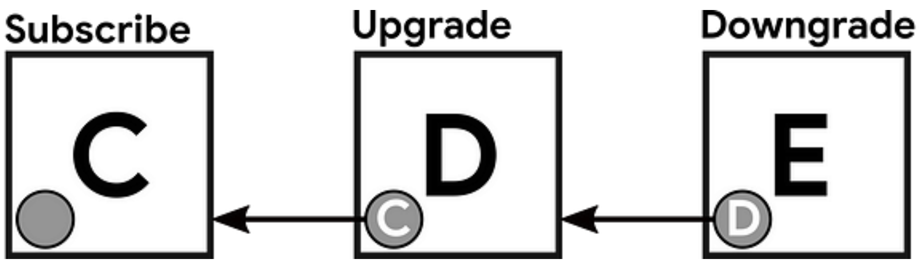
:er upgrades their subscription, a new purchase token B will be
ed. Since the upgrade is replacing the subscription from purchase
, the **linkedPurchaseToken** field for token B (shown in the grey
vill be set to point to token A. Notice it points backwards in time to
inal purchase token.



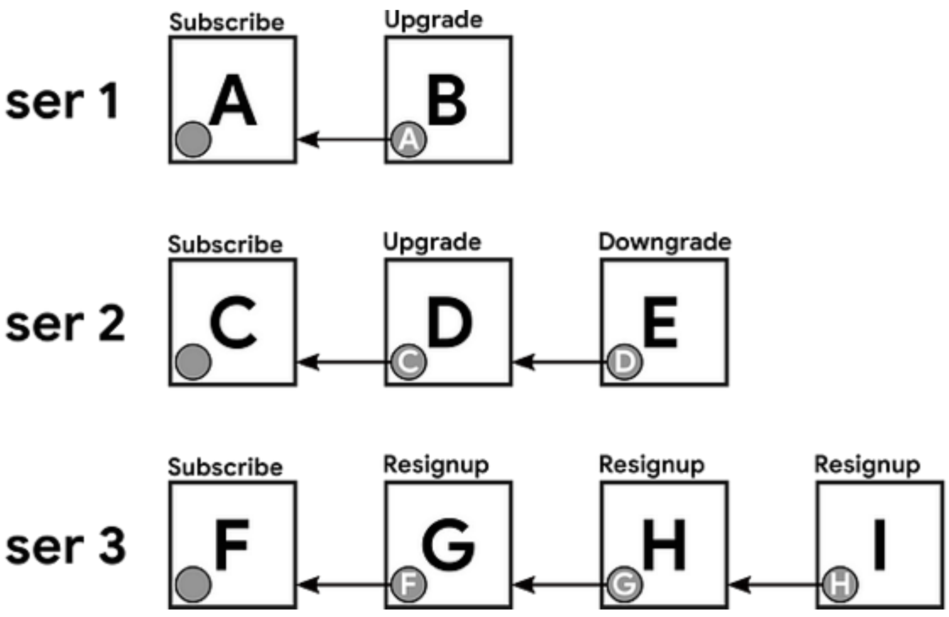
:e token B will be the only token that renews. Purchase token A ıot be used to grant users access to your content.

the time of upgrade, both purchase token A and B will indicate they ʋe if you query the Google Play Billing server. We will talk about this the next section.

:'s suppose a different user performs the following actions: subscribe, ɛ, downgrade. The original subscription will create purchase token C, rade will create purchase token D, and the downgrade will create ɛe token E. Each new token will link backward in time to the previous



d a third user to the example. This user keeps changing their mind. initial subscription, the user cancels and re-subscribes (does a p) three times in a row. The initial subscription will create purchase and the resignups create G, H, and I. The purchase token I is the cent token.



st recent tokens–B, E, and I–represent the subscriptions that users 1, , respectively, are entitled to and paying for. Only these most recent ıre valid for entitlement. However, all of the tokens in the chain are ıs far as Google Play is concerned, if the initial expiry date has not yet

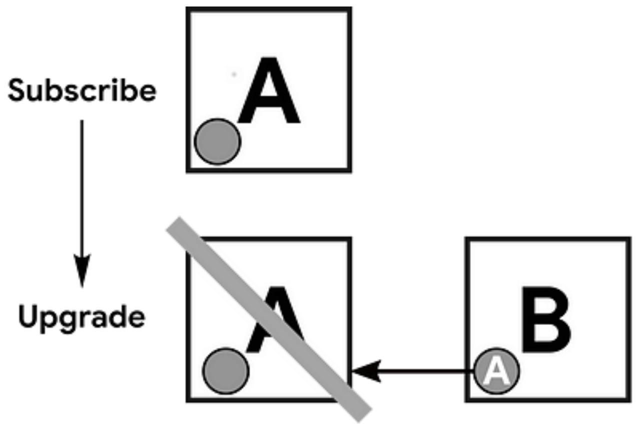
words, if you query the Subscriptions Get API for any of the tokens, ıg A, C, D, F, G, or H in the diagram above, you will get a Subscription e Response that indicates that the subscription has not expired and payment has been received, even though you should only grant ıent for the latest tokens.

y seem odd at first: why would the original tokens appear to be valid er they have been upgraded? The short answer is that this entation offers developers more flexibility when providing content vices to their users and helps Google protect user privacy. However, it quire you to do some important bookkeeping on your back-end

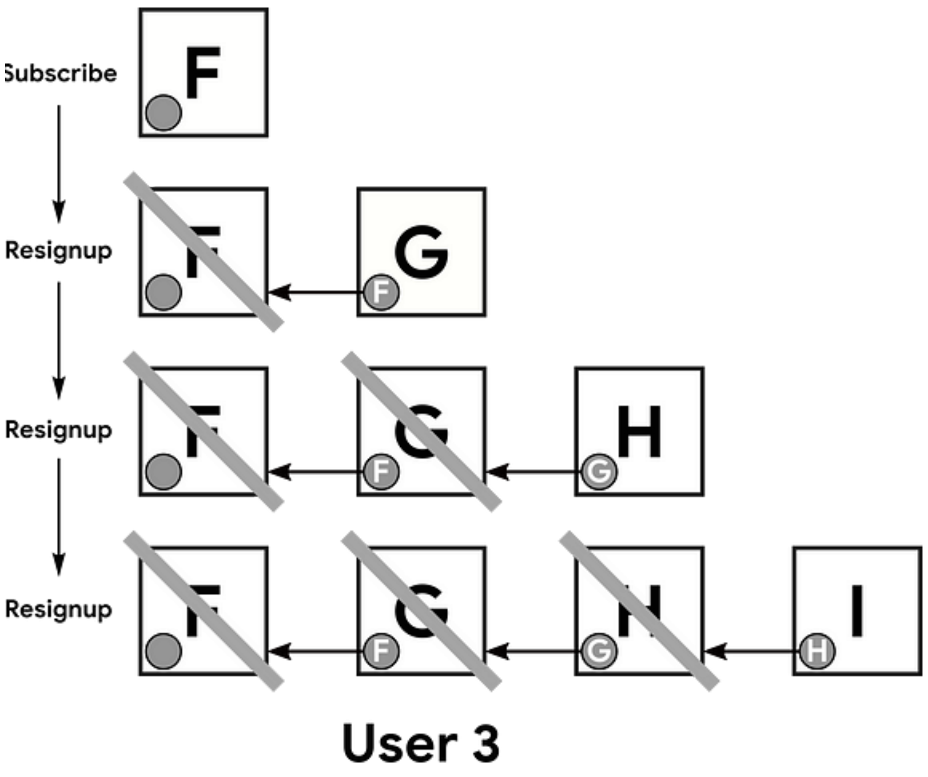
ng linkedPurchaseToken

me you verify a subscription, your back-end should check if the **urchaseToken** field is set. If it is, the value in that field represents the s token that has now been replaced. You should immediately mark vious token as invalid so that users cannot use it to access your .

ser 1 in the example above, when the back-end receives the purchase for the initial purchase, with an empty **linkedPurchaseToken** field, it entitlement for that token. Later, when the back-end receives the rchase token B after the upgrade, it checks the **linkedPurchaseToken** es that it is set to A, and disables entitlement for purchase token A.



vay, the back-end database is always kept up-to-date with which e tokens are valid for entitlement. In the case of User 3, the state of base would evolve as follows:



code for checking **linkedPurchaseToken:**

see an example of this in the Firebase back-end of Classy Taxi, an
urce end-to-end subscription app. Specifically, see the
ReplacedSubscription method in PurchasesManager.ts.

up an existing database

ur back-end will be kept up-to-date with new, incoming purchase
you will check each new purchase for the **linkedPurchaseToken**
d any tokens corresponding to a replaced subscription will be
y disabled. Awesome!

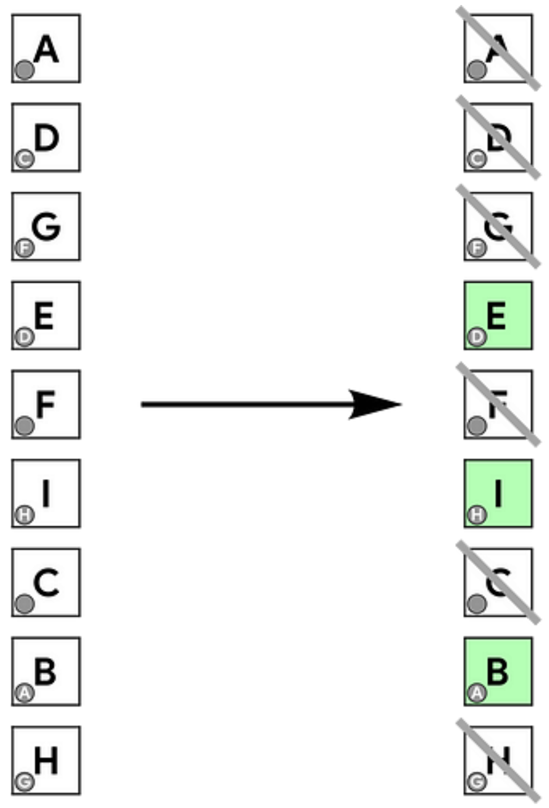
t if you have an existing database of subscriptions which did not
for the **linkedPurchaseToken** field? You will need to run a one-time
algorithm on your existing database.

cases, the most important thing when cleaning up a database is
or not a given token is entitled to content/services. In other words:
ot be necessary to recreate the upgrade/downgrade/resignup
e history for each subscription, only to determine the correct
nent for each individual token. A one-time clean-up of the database
things into shape and, moving forward, new incoming subscriptions
d to be handled as described in the previous section.

the purchase tokens for our three users above are stored in a
e. These purchases may have happened over time and could appear
rder. If the clean-up function does this right, tokens B, E, and I
end up marked as valid for entitlement and all the other tokens

›e disabled.

ː time through the database and check each element. If the **urchaseToken** field is set, then disable the token contained in that ›r the diagram below, we move through from top to bottom:



```
ıt A: linkedPurchaseToken not set, move to next
ıt D: linkedPurchaseToken == C, disable C
ıt G: linkedPurchaseToken == F, disable F
ıt E: linkedPurchaseToken == D, disable D
ıt F: linkedPurchaseToken not set, move to next
```

code for cleaning-up existing database:

nning this one-time clean up, all the old tokens will be disabled and :abase will be ready to go.

;ecurity

.er help protect against suspicious activity, it is also a good idea to set **ountId** field in your app using the BillingFlowParams.Builder's untId method. You should set this to a queryable value that is unique user but that obfuscates any user data, like a one-way secure hash of :'s account name.

: but important

ıt you understand how the **linkedPurchaseToken** field works, make
handle it correctly in your back-end. Every app with subscriptions
oe checking this field. Correctly keeping track of entitlement is
to ensuring the right user is granted the right entitlement at the right

rces

gle Play Billing Library

scription upgrades and downgrades

scriptions API

syTaxi end-to-end subscriptions sample app

up *refers to when a user subscribes, cancels their subscription, and then re-*
es before the original subscription has expired. Although they have not lost
ent and the new subscription will be the same as the previous one, they
hrough another purchase flow as they are committing to future payments.
l receive a new purchase token and the linkedPurchaseToken field will be
the case of an upgrade or downgrade. Update: note, this occurs for a
from within an application only. If a user re-subscribes from the Google
re Subscription Center, a new purchase token will not be issued and this
not be set — the original token will be used.

Linkedpurchasetoken Google Play Subscription Android

249 | 5 |