Price Change Runbook

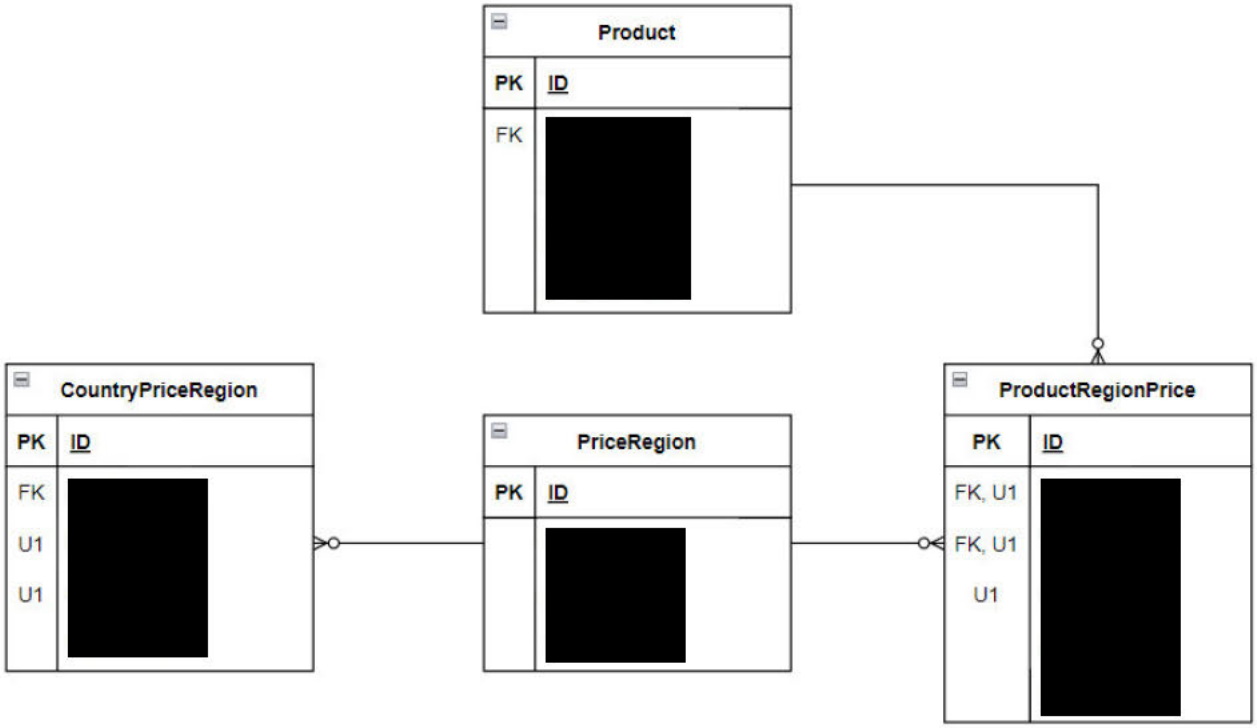
What it does, in one sentence

This runbook describes how the Subscriptions team makes price changes, handles legacy pricing, and cancels subscriptions across first-party and third-party services.

Overview

First-Party Product Pricing

The following schema manages our first-party product pricing and is stored in the [redacted] DB within the [redacted] RDS:



A **PriceRegion** represents a list of country codes that behave similarly together and share a currency code, such as "United States of America" or "Eurozone."

In the **CountryPriceRegion** table, each country code is mapped to a **PriceRegion**. Whichever row has the highest version with an effective date in the past is used. If no country code entry exists on the **CountryPriceRegion** table, we use a fallback country code of `ZZ`. The **PriceRegion** contains the currency code for all prices mapped to that region.

For a given **Product** and **PriceRegion**, the price amount is determined by the **ProductRegionPrice** table. The row with the highest version with an effective date in the past is used. If no entry exists, then no price is currently set for that product and region.

Third-Party Product Pricing

Third-party product pricing is handled in each respective administration tool.

For price change information on each third-party product go to:

- iTunes - App Store Connect: Manage pricing for auto-renewable subscriptions
- Google Play - Changing subscription prices
- Roku - Change pricing for in-channel subscriptions

Taxes

- Users who have their prices changed must pay appropriate taxes on the SKU they are subscribed to.
- Tax may either be inclusive or exclusive to the base SKU cost.
- Typically, the Payments team handles any tax rate changes.
    - Taxes are handled by each payment processor. When an invoice is created, it passes a `tax included` flag to SPP to denote whether or not the invoice price includes tax.

For more information on how [redacted] handles taxes, go to Subscriptions and Payments Taxes.

Procedures

This section describes price change scripts to be run by the subscriptions team. For an example of cross-team price change procedures, go to Price Decreases Phase 1 Playbook - [redacted].

The following set of scripts update existing first-party subscription product prices. The scripts will:

- Generate the input file that contains the existing information about the product prices for the [redacted] tenant user ID you want.
- Update the select[redacted] tenant with the changes you want to make to the subscription prices

After you have generated the information to update subscription prices, DevOps will assist with running the scripts, and any output files must be sent to the subscriptions team.

> At this time, all changes are manually completed using scripts run in the deployed environment, such as `prod.`

Scheduling a Price Change for New Subscriptions

The following procedure describes how to change prices for new users.

Scheduling a Price Change

1. Create a .csv file with the new prices. It must contain the following columns in this exact order:

```
country code
old currency code
```

```
new currency code
premium 1 month amount
premium 3 month amount
premium 1 year amount
[redacted] 1 month amount
[redacted] 1 year amount
[redacted] 1 month amount
effective date
```

2. Within the app root of the `subscription-processor` repository, run the following:

[redacted]

[redacted]`version` must be an integer.
Recommended: The date when the price change requirements were received (in case it changes before launch).

3. Make sure there are no errors.
4. Using the output file, create a Flyway file in `db/`[redacted].
   Version it accordingly (update the minor version, not the major version, for seed updates).
5. Test locally before deploying to [redacted].

Verifying the Price Change

- This convenience script verifies the prices returned by the API.
- This script can be run both before and after applying the price change to see the price difference.
- Using the same input file as when you're scheduling new prices, run the following in the app root of the subscription-processor repository:

```
go run
 api ho
 tenant
 input
```

[redacted]

You can change `api-host` to any of the deployed environments' URLs.

Updating Legacy Prices

This procedure describes how we change prices for existing subscribers. These users keep their legacy prices for a certain period before we raise their subscription price to the new baseline rate.

- All current prices are considered legacy prices by default. This is because the system uses the price stored in the **Subscription** and **SubscriptionProduct** rows (currency code + amount).
- There is currently no automated way for the system to update the prices for users on legacy pricing.
    - To update prices for existing subscribers, the corresponding **SubscriptionProduct** rows need to be updated with the database script described below.

Updating Prices for Existing Users

1. In the [redacted] **db**, run the following script to generate the input file.
   The expected format is one legacy [redacted] user ID (numerical) per line.

```
SELECT s.account id from c[redacted].subscription as s
LEFT JOIN c[redacted].subscription product as sp on s.id   sp.subscription id
WHERE s.is active   1
  AND s.country code in ('2 LETTER COUNTRY CODE', ...)
  AND sp.product id <> 1337;    not employee sku
```

2. Run the following in the deployed environment:

```
[redacted]
```

   This script can run several concurrent processes with different input files.
3. Upload the output file to the associated DevOps Jira ticket for auditing.
   For an example ticket, go to [redacted] Run Script for Phase 2 Price Changes.

Canceling Subscriptions

This procedure describes how to cancel subscriptions for users who either opted out or didn't opt in before a given date.

Considerations

The Marketing team must provide a .csv file containing information for users who opted in to the price change. It must contain the following columns in this exact order:

```
account id
country
price change active products
tier
next renewal date
external id
email
price increase renewal date
```

Canceling Subscriptions

1. Generate the list of subscribers located in opt-in and opt-out countries using the following queries:

   You'll need to edit the following queries based on the countries and user cohorts affected by the price change.

   Opt-out Fan and Mega Fan (Monthly, Quarterly)

```
SELECT account id, country code, sku, T.name as tier, next renewal date
FROM [redacted] subs.subscription product SP
LEFT JOIN [redacted] subs.subscription S
   ON SP.subscription id = S.id
LEFT JOIN [redacted] subs.product P
   ON SP.product id = P.id
LEFT JOIN [redacted] subs.tier T
   ON P.tier id = T.id
WHERE country code IN [redacted]
AND is active = 1
AND P.sku IN [redacted]
[redacted]1 month')
AND product id != 1337;
```

   Opt-in Fan and MegaFan (Monthly, Quarterly, Yearly Renewal between 10.31-11.30)

```
SELECT account id, country code, sku, T.name as tier, next renewal date
FROM cr subs.subscription product SP
LEFT JOIN [redacted] subs.subscription S
   ON SP.subscription id = S.id
LEFT JOIN [redacted] subs.product P
   ON SP.product id = P.id
LEFT JOIN [redacted] subs.tier T
   ON P.tier id = T.id
WHERE country code IN [redacted]
AND is active = 1
AND P.sku IN ('[redacted] premium.1 month', '[redacted] premium.3 month',
'[redacted] [redacted]1 month') OR (P.sku IN ('[redacted] premium.1 year',
```

```
'[redacted]1 year') AND next renewal date BETWEEN '2022-10-31
14:00:00' AND '2022-11-30 23:59:59'))
AND product id != 1337;
```

2. Cross reference each of the queries above with the following query, ensuring that the respective country codes match:

```
SELECT account id, country code, sku, T.name as tier, next renewal date
FROM [redacted].subscription S
    LEFT JOIN [redacted].future subscription changes FSC
    ON S.id   FSC.subscription id
LEFT JOIN [redacted].product P
    ON FSC.future product id   P.id
LEFT JOIN [redacted].tier T
    ON P.tier id   T.id
WHERE country code IN [redacted]
AND is active   1
AND next renewal date <  '2022 11 30'
AND FSC.future product id IN (4, 5, 6, 10, 2000)
AND FSC.subscription id is not null
AND FSC.status   0
```

3. Using the .csv file you generated, and the .csv file provided by Marketing, run the following script within the app root of the `subscription-processor` repository:

```
go run bin/price change input.go
 marketing export input csv placeholder marketing export.csv
 sent input csv placeholder [redacted]file.csv
 price output csv price change output.csv
```

4. Split the output file into multiple files with the following script. Make sure the header row is not changed or deleted, as the cancellation script relies on it to function.

> If you're running this on Mac OSX, use `gsplit` instead of `split`. You may have to call: `brew install coreutils`

```
export inputPrefix 'price change output' parts 16 && split   verbose  d  n l/$
{parts}
 additional suffix .csv
 filter '([ "$FILE" !  "${inputPrefix}.00.csv"   && head  1 "$
{inputPrefix}.csv" ; cat) > "$FILE"' "${inputPrefix}.csv" "${inputPrefix}."
```

5. Coordinate with DevOps to run the following files on the [redacted]`proration` machine in the `prod` environment:

```
[redacted] proration/maintenance
```

```
config file /srv/[redacted] proration/parameters.yml
command [redacted] opt in cancel
[redacted] subs base url http://[redacted].com
[redacted] list given filepath {filename}
```

For an example DevOps ticket, go to OPS-19091: Run Script for Phase 2 Price Changes.

Updating the tax_included flag

The list of `tax included` country codes are maintained in the payments api.go file of the subscription-processor repository.
Add to or remove from the list as necessary. This requires a deployment.

More Information

Subscriptions and Payments Subscription Tiers

Price Changes - Record of Progress