UX Writing Portfolio
@ Google Stadia (via Adecco)

Austin Fisher

About Me

Austin Fisher

I’m a storyteller, and I strive to create experiences that make people **feel**. I use plain (& fun) language to explain tough concepts, and use design thinking to solve unconventional problems.

I want to make a positive impact on the world by making people’s lives easier and more enjoyable, and I think UX is the best way to accomplish that.

All it takes is the right word.

My UX Writing Principles

Austin Fisher

UX writing is all about helping people complete their tasks with confidence.
Regardless of what I'm writing, there are a few principles I always try to follow.

Be **Helpful**

Users always have a mission. Help them succeed faster.

This can be linking to FAQs, recommending a relevant product, or sorting results to be more valuable. Let's make their job easier.

Be **Brief**

People don't read. Keep things short and scannable.

We process information way easier in bite-sized chunks.

Be **Consistent**

Why re-invent the wheel? Use familiar industry terms, and re-use common words in similar buttons.

Consistency sets a pattern. This pattern helps users navigate the product much faster.

Be **Mindful**

Always read the room. There's a time and place for all content.

Sympathize when something goes wrong, celebrate when things go right, and be direct when communicating errors.

Be **Human**

Being human is more than using contractions and clever taglines.

Use human terms, talk with the user (not at them), and make them feel supported throughout the experience.

Be **Creative**

Technical doesn't have to mean boring. Use metaphors to describe something complex, or use examples and images to explain difficult concepts.

A picture's worth 1000 words.

Austin Fisher

Error Strategy

Summary

I was hired as a UX Writer for Stadia because partners repeatedly told us:

> “If you’re going to fix anything, make it your errors. I spend hours trying to figure out problems that take seconds to resolve.”

> “Definitely improve your errors. I don’t know many times I’ve hit ‘internal error encountered’ and gave up.”

Needless to say, I’ve dont **lots** of error writing.

In this section, you’ll see:
- Case study: File validation & 1000+ errors
- How to write an error
- Error writing samples

Error Strategy

Austin Fisher

Case study: File validation & 1000+ errors

End-to-end strategy and error writing

Error Strategy: File validation & 1000+ errors

Austin Fisher

Problem

Uploading a storefront file produced thousands of errors, many of which were repetitive, unhelpful, inaccurate, or convoluted, making it nearly impossible to create a valid file.

This blocked partners from submitting their game to Stadia, and required manual intervention by our internal Operations team, costing both Stadia and our partners **100s** of hours of time.

Solution

Perform a full audit of all errors and canonical codes used, identify gaps, and use that data to create a design framework. Then, rewrite all of the errors to meet content standards.

My Role

I led this feature from start to finish. Here are the key tasks I accomplished during this project:

- Learned how to read the codebase to understand how certain errors are thrown.
- Became an expert in canonical codes and how to spot gaps in error messaging.
- Performed a full audit of all the possible errors for both the Excel and JSON versions.
- Outlined a 2-phase approach for tackling these errors, collaborating with PM and engineering.
- Organized the errors in a way that made it easy for engineering to implement.
- Rewrote all the errors, identifying patterns and templates to simplify the coding process.
- Created a design strategy for displaying the errors.

Step 1: Identify the problem

Imagine you're putting together some images for your game's store and naming the achievements. You upload the first version of the storefront, and you see... 5003 errors.

Validation results

The file contains 5003 errors. All the errors should be fixed, otherwise the storefront may fail certification.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[0].image.variants['001_de-DE'].image.locators_1x[0]' can't be 'Size: 960 x 540 px' because it's not an accepted value. Update this field to one of the following: Size: 928 x 522 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[0].image.variants['001_de-DE'].image.locators_1.5x[0]' can't be 'Size: 1440 x 810 px' because it's not an accepted value. Update this field to one of the following: Size: 1392 x 783 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[0].image.variants['001_de-DE'].image.locators_2x[0]' can't be 'Size: 1920 x 1080 px' because it's not an accepted value. Update this field to one of the following: Size: 1856 x 1044 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[0].image.variants['001_de-DE'].image.locators_3x[0]' can't be 'Size: 2880 x 1620 px' because it's not an accepted value. Update this field to one of the following: Size: 2784 x 1566 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[1].image.variants['001_de-DE'].image.locators_1x[0]' can't be 'Size: 960 x 540 px' because it's not an accepted value. Update this field to one of the following: Size: 928 x 522 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[1].image.variants['001_de-DE'].image.locators_1.5x[0]' can't be 'Size: 1440 x 810 px' because it's not an accepted value. Update this field to one of the following: Size: 1392 x 783 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[1].image.variants['001_de-DE'].image.locators_2x[0]' can't be 'Size: 1920 x 1080 px' because it's not an accepted value. Update this field to one of the following: Size: 1856 x 1044 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[1].image.variants['001_de-DE'].image.locators_3x[0]' can't be 'Size: 2880 x 1620 px' because it's not an accepted value. Update this field to one of the following: Size: 2784 x 1566 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[2].image.variants['001_de-DE'].image.locators_1x[0]' can't be 'Size: 960 x 540 px' because it's not an accepted value. Update this field to one of the following: Size: 928 x 522 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[2].image.variants['001_de-DE'].image.locators_1.5x[0]' can't be 'Size: 1440 x 810 px' because it's not an accepted value. Update this field to one of the following: Size: 1392 x 783 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[2].image.variants['001_de-DE'].image.locators_2x[0]' can't be 'Size: 1920 x 1080 px' because it's not an accepted value. Update this field to one of the following: Size: 1856 x 1044 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[2].image.variants['001_de-DE'].image.locators_3x[0]' can't be 'Size: 2880 x 1620 px' because it's not an accepted value. Update this field to one of the following: Size: 2784 x 1566 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[3].image.variants['001_de-DE'].image.locators_1x[0]' can't be 'Size: 960 x 540 px' because it's not an accepted value. Update this field to one of the following: Size: 928 x 522 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[3].image.variants['001_de-DE'].image.locators_1.5x[0]' can't be 'Size: 1440 x 810 px' because it's not an accepted value. Update this field to one of the following: Size: 1392 x 783 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[3].image.variants['001_de-DE'].image.locators_2x[0]' can't be 'Size: 1920 x 1080 px' because it's not an accepted value. Update this field to one of the following: Size: 1856 x 1044 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[3].image.variants['001_de-DE'].image.locators_3x[0]' can't be 'Size: 2880 x 1620 px' because it's not an accepted value. Update this field to one of the following: Size: 2784 x 1566 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[4].image.variants['001_de-DE'].image.locators_1x[0]' can't be 'Size: 960 x 540 px' because it's not an accepted value. Update this field to one of the following: Size: 928 x 522 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[4].image.variants['001_de-DE'].image.locators_1.5x[0]' can't be 'Size: 1440 x 810 px' because it's not an accepted value. Update this field to one of the following: Size: 1392 x 783 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[4].image.variants['001_de-DE'].image.locators_2x[0]' can't be 'Size: 1920 x 1080 px' because it's not an accepted value. Update this field to one of the following: Size: 1856 x 1044 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[4].image.variants['001_de-DE'].image.locators_3x[0]' can't be 'Size: 2880 x 1620 px' because it's not an accepted value. Update this field to one of the following: Size: 2784 x 1566 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[5].image.variants['001_en-US'].image.locators_1x[0]' can't be 'Size: 960 x 540 px' because it's not an accepted value. Update this field to one of the following: Size: 928 x 522 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[5].image.variants['001_en-US'].image.locators_1.5x[0]' can't be 'Size: 1440 x 810 px' because it's not an accepted value. Update this field to one of the following: Size: 1392 x 783 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[5].image.variants['001_en-US'].image.locators_2x[0]' can't be 'Size: 1920 x 1080 px' because it's not an accepted value. Update this field to one of the following: Size: 1856 x 1044 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[5].image.variants['001_en-US'].image.locators_3x[0]' can't be 'Size: 2880 x 1620 px' because it's not an accepted value. Update this field to one of the following: Size: 2784 x 1566 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[6].image.variants['001_en-US'].image.locators_1x[0]' can't be 'Size: 960 x 540 px' because it's not an accepted value. Update this field to one of the following: Size: 928 x 522 px.

In product 'ghost-skin', the field 'media_assets.featured_16x9s[6].image.variants['001_en-US'].image.locators_1.5x[0]' can't be 'Size: 1440 x 810 px' because it's not an accepted value. Update this field to one of the following: Size: 1392 x 783 px.

How does this wall of text make you feel? Is it obvious how to solve all these errors? How long would it take to solve everything?

Now look at this:

Validation results

The file contains 86 errors. All the errors should be fixed, otherwise the storefront may fail certification.

This storefront contains 2 visible gamer stats, but it's required to have at least 3 visible. Update the field 'visibility' to 'PROFILE' for at least 1 gamer stat.

The following sections are missing:
- video_config
- audio_config
- stream_connect_config

There are 12 product IDs from the published storefront that are missing in this version. Products can't be removed from the game once they're published. Add the following products back to the storefront:
- worm-day-subscription
- worm-year-subscription
- sparkly-color-skin
- ghost-skin
- angry-color-skin
- deluxe
- cow-skin
- health-boost
- rainbow-color-skin
- polka-dot-skin
- dragon-skin
- cat-color-skin
- featured_16x9s.display_order (the value '1s' must be an integer.)

In product 'game', the field 'media_assets' is missing 15 fields. Add the following fields:
- media
- media_small
- featured_16x9s
- featured_1x1s
- featured_thumb_16x9s
- branded_keyart_16x9
- branded_keyart_1x1
- branded_keyart_3x4
- branded_keyart_4x3
- branded_keyart_banner_3x1
- branded_keyart_small_16x9
- branded_keyart_thumb_16x9
- branded_keyart_thumb_1x1
- branded_keyart_thumb_4x3
- branded_keyart_thumb_small_1x1

In product 'game', the field 'key_art_color' has a very low contrast ratio of '1.65:1' with the Stadia gray text (#A8ADB3). The contrast ratio must be at least 4.5:1, but should be 7:1 to meet AAA accessibility standards. Provide a different color.

It's still a lot of errors, but doesn't it feel like you can breathe a bit easier? This was my approach to reducing thousands of errors into something manageable.

My priorities were:
- Rewrite to fix inaccuracies in the errors
- Bundle all similar errors together into one message
- Create stronger parsing logic to have more custom, human-readable errors.

Step 2: Understand the problem

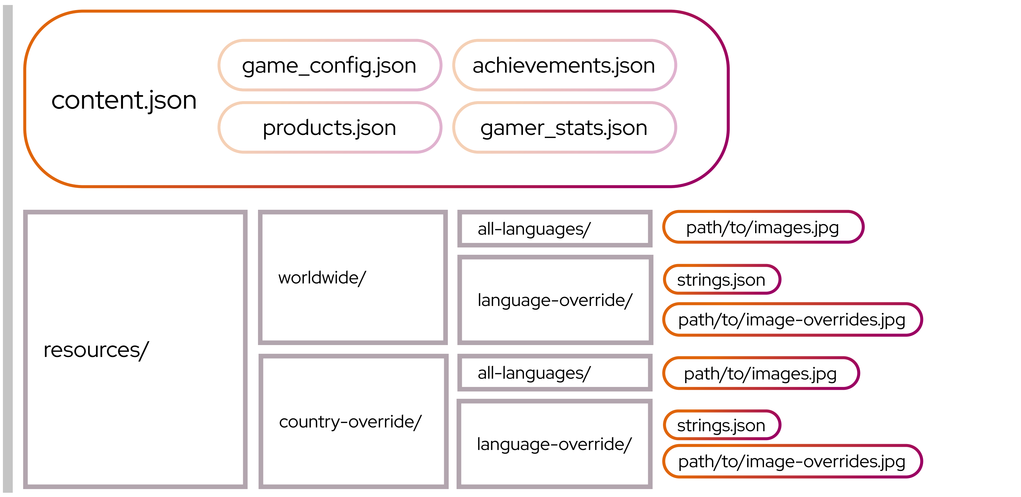
A storefront file contains a lot of pieces that can go wrong. It can be provided in a series of Excel files, or it can be uploaded as a big JSON file. Even though the data and images are the same, they're provided in very different ways depending on your upload preference.

Problem: The Excel and the JSON files don't map 1-to-1, and resources are stored very differently. This makes it difficult to pinpoint an error's source.

Structure of a JSON storefront

A JSON storefront prioritized all data in one place. The main 'content.json' file contained all this data. It was a goliath of a file though, 10,000 lines easily -- and the publisher had to manually type most of it themselves.

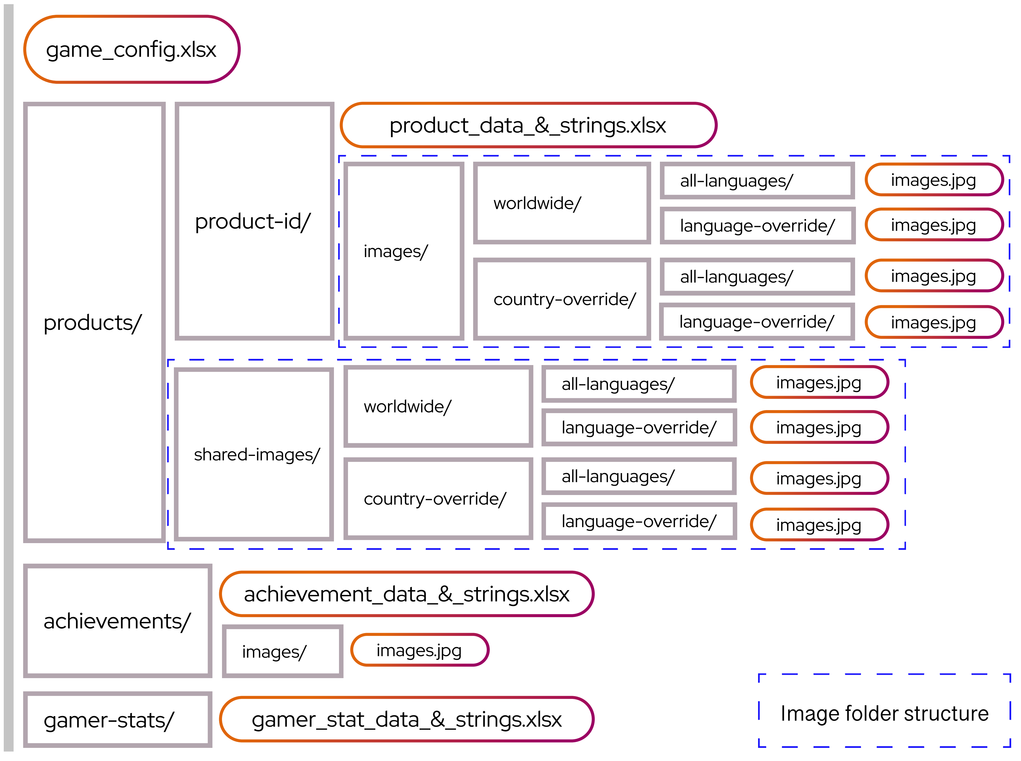
Publishers also had to manually construct the resource folders, which was very time-consuming and prone to error, especially since all the country/language combinations were mandatory.



Structure of an Excel storefront

The Excel storefront was an effort to make the daunting task of creating a storefront much easier to complete. By working entirely in spreadsheets, all the fields were easy to see and fill in, many were drop-downs with validation, and we could pinpoint the exact cell that had an error.

The Excel version introduced "shared images" to remove all duplicate images from the storefront. In the JSON version, images were duplicated for every product, which was very impractical and took a lot of space.



Prioritizing the work

We introduced the Excel sheet as a much simpler way to create a storefront, so our internal Operations team would no longer be required to assist every partner. We wanted to migrate all partners to the sheets.

But without proper error validation, the sheets weren't much more helpful than JSON, and our internal Operations team faced just as much stress.

So, I focused on writing all the Excel errors first. This would have the biggest impact, saving many hours for both our partners and our internal teams, and removing one of the biggest blockers to publishing a game.

Step 3: Designing a solution

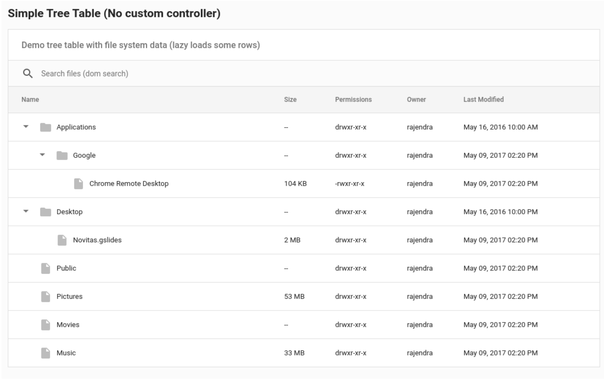
Creating a design template

Before writing errors, I needed to think about how they'd be presented to partners. I explored some design options, working closely with our Visual Designer to find a component that fit with our existing assets.

We discovered the **file tree table**, a pattern that was very similar to our existing tables, but could also expand and collapse for viewing files.

Pros:
- Adaptable for many situations across the product (e.g. replacing one file in a game's code)
- Good for locating invalid image files, with 1 or 2 errors each.

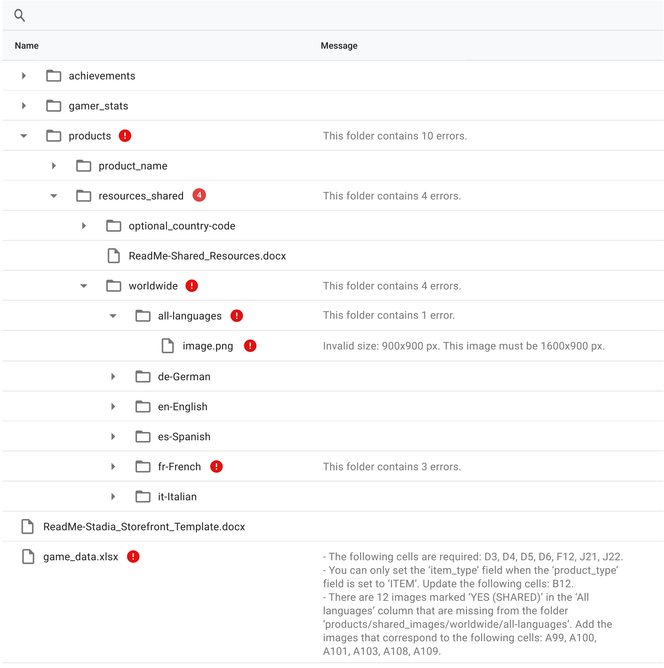
Cons:
- Not meant for large amounts of errors.
- Need to dig into a folder to find the error.
- Partners don't care about the valid files.



We explored how the file tree would adapt to our use case. It was great to see at-a-glance where the errors were before even reading anything, but it fell apart when validating spreadsheets.

How would we show 30 errors? 100 errors? Would we need a separate page?

Suddenly, this didn't seem to make sense anymore.



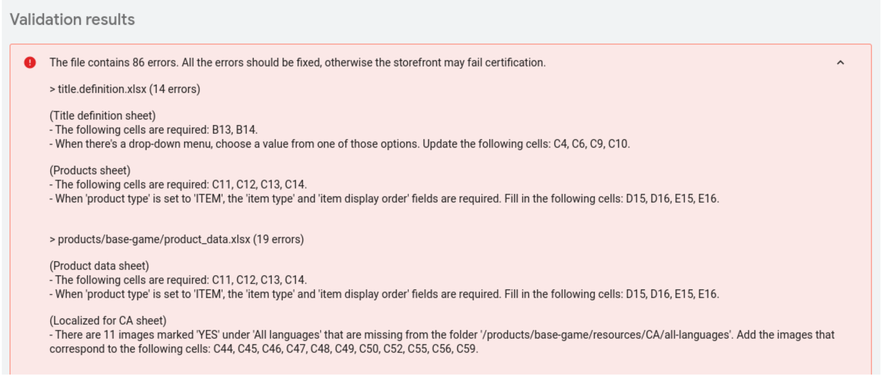
Designing with **one** string

At this point, engineering told us about a technical limitation: Our backend could only send one error string to the front end.

Problem: The error had to be delivered as one big string. We couldn't apply any front-end formatting.

Goal: Easily identify **what** the error is, and **where** to fix it.

Solution: Remember the big wall of errors from the start? I went back to that template to see what I could work with. We couldn't apply fancy formatting, however we **could** still add line breaks and apply basic text formatting in the string itself. So I got creative and drafted a way to organize the errors within our limitations.



I didn't want these errors to be repetitive. The best way to showcase similar errors was to bundle them together whenever possible. I grouped them by file path, then by specific sheet within the file, then by error type (required field, localization error).

I added white space to help distinguish all the different sheets and files.

Now that I had the design mapped out, I could focus on writing all the strings.

Step 4: Writing the Excel errors

Auditing all the errors

I did an audit of the spreadsheets to find all possible types of errors for every field. Most fields followed the same few error types, so many strings were reusable, like invalid values and required fields.

Error type	Error case	Structure	Example
REQUIRED_FIELD	Empty field.	"The following cells are required: [cells]."	"The following cells are required: B13, B14."
	Conditional. 1 field affected. On specific entry.	"When '[field1]' is set to '[value]', the '[field2]' field is required. Fill in the following cells: [cells]."	"When 'product type' is set to 'BUNDLE', the 'Bundle contents' field is required. Fill in the following cells: F13, F14, F15."
	Conditional. 1 field affected. On general entry.	"When '[field1]' is provided, the '[field2]' field is required. Fill in the following cells: [cells]."	"When 'Country' is provided, the 'Release date' field is required. Fill in the following cells: D42."
	Conditional. 2+ fields affected. On specific entry.	"When '[field1]' is set to '[value]', the '[field2]', '[field3]', [...] and '[field#]' fields are required. Fill in the following cells: [cells]"'	**(no known situation exists yet)**
REQUIRED_FIELD *WARNING*	Empty field. Warning.	"The following cells are empty and should be provided: [cells]."	"The following cells are empty and should be provided: B33, B35."
	Conditional. 1 field affected. On specific entry. Warning.	"When '[field1]' is set to '[value]', the '[field2]' field should also be provided. Fill in the following cells: [cells]."	"When 'product type' is set to 'GAME', the 'Game genre' field should also be provided. Fill in the following cells: B17."
	Conditional. 1 field affected. On any general entry. Warning.	"When '[field1]' is provided, the '[field2]' field should also be provided. Fill in the following cells: [cells]."	"When 'Description (locked)' is provided, the 'Achievement name (locked)' field should also be provided. Fill in the following cells: J12, J13, J14, Q12, Q13, Q14, X12, X13, X14."
	Conditional. 2+ fields affected. On specific entry. Warning.	"When '[field1]' is set to '[value]', the '[field2]', '[field3]', and '[field4]' fields should be provided. Fill in the following cells: [cells]"	"When 'product type' is set to 'GAME', the 'Privacy policy URL', 'Terms of service URL', 'Product website URL', and 'Publisher support URL' fields should be provided. Fill in the following cells: A12, A13, A14, A15."
INVALID	Already exists. An ID	"Can't define the same ID multiple	"Can't define the same ID multiple
VALUE (ALREADY EXISTS)	appears more than once.	times. Update or remove the duplicate rows of the following IDs: [IDs]."	times. Update or remove the duplicate rows of the following IDs: 'base-game', 'red-armor'."
INVALID VALUE	Must be from drop-down.	"There are # invalid values. You must choose a value from one of the dropdown options. Update the following cells: [cells]."	"There are 4 invalid values. You must choose a value from one of the dropdown options. Update the following cells: C4, C6, C9, C10."
	Conditional. 1 field affected. Empty.	"When '[field1]' is set to '[value]', the '[field2]' field should be empty. Update the following cells: [cells]."	"When 'Data type' is 'STRING', the 'Display format' field should be empty. Update the following cells: E16, E22."

Strategy

1) I filled in the first 2 columns with the type of error, and the specific scenario.

Whenever I discovered a new conditional issue ("when A is set to 1, B must be set to 2"), I logged it as a separate error scenario.

2) I wrote the error so it could list any number of cells. I made sure to use consistent wording so we could re-use the same template.

When writing, I spelled out all the variables we'd need to grab from the backend, so engineering could plan work accordingly.

3) I asked engineering if the new variables were feasible to grab, and rewrote when necessary.

4) I wrote an example of the error in action, plugging in all the variables to show how it would read in context.

5) I marked in the appendix which fields threw which errors. This way, engineering could see an exhaustive list of all errors for each spreadsheet.

6) I noted any custom errors that needed special attention, so engineering knew to create new error codes.

Appendix A: All possible Excel errors by file

Resources for finding all errors: Products, Achievements + Gamer stats, Multiplayer
Green-coded strings have general templates, blue-coded strings are customized templates, and orange-coded strings can't be executed by the storefront checker and must remain in the GamerX backend.

File	Sheet	Error case	String type / Custom string
Title_definition.xlsx	**(title def sheet)**	Required field. (Game engine, Dynamic range, Maximum stable framerate at 1080p, RESOLUTION_4K, CHANNEL_MODE_SURROUND51, Stream Connect, Queue to play, Crowd boost)	Empty field
		Not required, but should be provided. (Stadia Package ID)	Empty field. Warning.
		Invalid value.	Must be from drop-down.
		Stadia package ID incorrect	Custom. Stadia package ID incorrect
	(product sheet)	Required field. (Product ID, Product type)	Empty field
		Required field - conditional on type being ITEM:(Item type) or BUNDLE: (Bundle contents)	Conditional. 1 field affected. On specific entry.
		Not required field but should be provided - conditional on others. (Item display order)	Conditional. 1 field affected. On specific entry. Warning.
		Product ID doesn't match a folder.	Missing product folder.
		Already exists - product ID	Already exists. An ID/field appears more than once.
		Invalid value.	Must be from drop-down
		Invalid field - not a positive integer (item display order)	Number. Must be a positive integer.
		Bundle. Product ID in bundle doesn't exist	Custom. Product ID in bundle
			incorrect
		Bundle. Bundle can't reference itself.	BUNDLE_INVALID_CONTENT_TYPE
		Bundle. Bundle must include game product ID.	BUNDLE_INVALID_GAME_COUNT
Product_data.xlsx	**(product definition)**	Required field. (Product ID, product type, Disclosures, Input supports, Accessibility features, Platform features, Gameplay modes, Accessible regions, Languages supported in-game, ESRB/PEGI/USK age ratings)	Empty field
		Not required, but should be provided. (Key art color, Min#/Max# of players, NON-GAME: Original release date)	Empty field. Warning.

Conclusion: Passing it off

After putting together every error possible, I knew it was ready to hand off. I got great feedback from engineering about the handoff process.

> "The errors were very clear and well-documented. We could easily identify the required logic. You made our jobs way easier by doing this."

The next steps: create alternate errors for the JSON validation.

Step 5: Writing the JSON errors

Organizing the errors

The JSON validation errors were similar... but not the same. Since we couldn't identify them by cell number, I had to change the writing approach to identify them by field names.

I used Google Sheets to organize all the errors this time. It was much easier to create templates and tie them together with formulas. I broke each message down into 5 parts:

Location: This is the entity or field that triggered the error. This affects the rest of the message.

Issue: The actual error.

Rules: The conditions that failed, or any other context like formatting.

Solution: How to fix the error. This usually lists the fields to update.

Help: Links to our documentation or to our partner support contact page.

Introduction (locating the error)		
	Single	**Bundle**
Entity-level	The [entity] 'id'	There are # [entities]
Field-level	In [entity] 'id', the field 'name'	In [entity] 'id', # fields

Issue	Context (rules)	Solution (action)	Help
is missing.	It's required when 'field2' is set.		
is missing.	It's required when 'field2' is set to 'VALUE'.		
are missing.	They're required when 'field2' is set.	Add the following fields: - field1 - field2 - field3	
are missing.	They're required when 'field2' is set to 'VALUE'.	Add the following fields: - field1 - field2 - field3	
has an invalid value of 'VALUE0'.		Update this field to one of the following values: 'VALUE1', VALUE2', 'VALUE3'.	
has an invalid value of 'VALUE0'.		Update this field to one of the valid values listed at: developers.google.com/stadia/docs/list-of-values	
has an invalid date of 'VALUE0'.	All dates must follow ISO 8601 format: 'YYYY-MM-DDThh:mm:ssZ'.		Learn more
has an invalid value of #.	This field must be a positive integer.		
has an invalid value of #.	The maximum allowed value is #.		

I recorded the canonical code, the type of error (which determined the location), and grouped similar errors together. This way it was obvious which errors shared the same messaging.

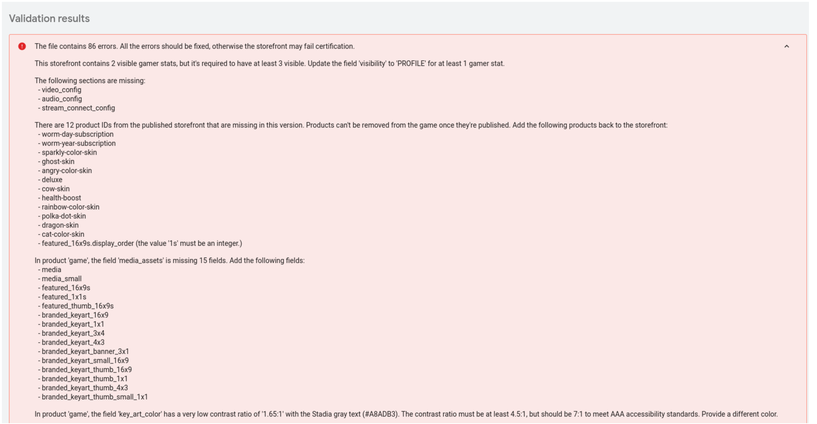
I tied all the fragments into the 'Final string', so I could read it over in context. Sometimes I'd combine fragments together that didn't make sense, so this helped me edit them to be more human.

Error code	Single / Bundle	Level of error	Scenario	Issue	Context (rules)	Solution (action)	Help	Final string
(NEW) INVALID_VALUE_CONTRAST	Single	Field	**Note: the field should match what was input, so 'theme_color', not 'key_art_color'.** **ERROR**	has a very low contrast ratio of # with the Stadia gray text (#A8ADB3).	The contrast ratio must be at least 4.5:1, but should be 7:1 to meet AAA accessibility standards.	Provide a different color.		In [entity] 'id', the field 'name' has a very low contrast ratio of #:# with the Stadia gray text (#A8ADB3). The contrast ratio must be at least 4.5:1, but should be 7:1 to meet AAA accessibility standards. Provide a different color.
	Single	Field	**WARNING**	has a low contrast ratio of #:# with the Stadia gray text (#A8ADB3).	The contrast ratio should be 7:1 to meet AAA accessibility standards.	Provide a different color.		In [entity] 'id', the field 'name' has a low contrast ratio of #:# with the Stadia gray text (#A8ADB3). The contrast ratio should be 7:1 to meet AAA accessibility standards. Provide a different color.
PRODUCT_DELETED + ACHIEVEMENT_DELETED	Custom		Single	The [entity] 'id' is in the published storefront, but is missing in this version.	[Entities] can't be removed from the game once they're published.	Add the [entity] 'id' back to the storefront.		The [entity] 'id' is in the published storefront, but is missing in this version. [Entities] can't be removed from the game once they're published. Add the [entity] 'id' back to the storefront.
+ GAMER_STAT_DELETED	Custom		Bundle	There are # [entity] IDs from the published storefront that are missing in this version.	[Entities] can't be removed from the game once they're published.	Add the following [entities] back to the storefront: - entity1 - entity2 - entity3		There are # [entity] IDs from the published storefront that are missing in this version. [Entities] can't be removed from the game once they're published. Add the following [entities] back to the storefront: - entity1
PRODUCT_TYPE_MISMATCH + PRODUCT_ITEM_TYPE_MISMATCH	Single	Field	All 4 type mismatches can share the same sentence with variables.	is 'VALUE1', which is different from the published storefront, where it's 'VALUE2'.	The field 'name' can't be modified after the [entity] is published.	Revert this field back to 'VALUE2'.		In [entity] 'id', the field 'name' is 'VALUE1', which is different from the published storefront, where it's 'VALUE2'. The field 'name' can't be modified after the [entity] is published. Revert this field back to 'VALUE2'.

Designing for **one** string... again

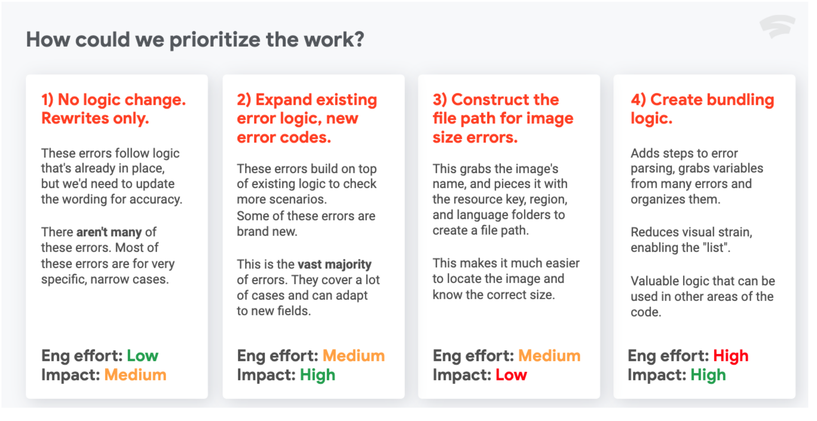
Problem: The design template for Excel cells didn't adapt well to long field names. It was much harder to tell where field names began and ended.

Solution: List the fields vertically to easily compare them all. Engineering said this isn't much of an issue.



Proposing features & passing it off (in progress)

I'm in the process of passing off this phase of the project. It's actually much tougher to edit existing error logic than it is to create it all from scratch, like we did with the Excel errors, so I broke it down into pieces when presenting to engineering and the PM.



The meeting went well, and now we're prioritizing the work. This is currently in progress! I'm working with engineering to get more accurate estimates about effort.

Next steps

The Excel errors from Step 4 were just implemented to our internal Operations team, so in the coming weeks, I should start getting feedback about what's working and what isn't. I may need to edit some errors or add in new cases.

Then, it'll finally be implemented for partners to use. At that point, the impact of this work will finally pay off!\

That's all for this case... for now.

Austin Fisher

How to write an error

Structure and examples

How to write an error

Errors are key to fixing problems and continue workflows. A bad error disrupts workflows for hours, even days. A good error clearly outlines the problem and solution so it only disrupts workflows for a few minutes. A great error is when the error isn't thrown at all, and the experience prevents it from happening altogether.

Structure of an error

Problem

Every error starts because of a problem. This must identify the **action** that went wrong, and the **item** that's affected.
At Stadia, we tend to start internal errors with "Unable to...", and user-caused errors with "Can't...".

Example: "Can't upload game to the project [name]."

Rules / context

There's a reason why this problem happened. Explain the rules: "when item A is true, item B is required."
This way, users know the cause and hopefully never make the error again.

Example: "The game is 40GB, but the project only has 25GB of cloud storage left."

Solution

The solution must identify an **action** to do, and the **item** to act on. Sometimes the solution is clear from the rules, or vice-versa.
Be explicit with the solution, but make sure it's adaptable to all endpoints. Instead of "Click an option", say "Specify an option".

Example: "Archive old game uploads to free up space, or ask an admin to assign more storage to this project."

Help

Sometimes an error is highly complex and can't be explained in 1 or 2 sentences. In that case, we may direct the user to our documentation, or ask them to contact us for support.

Example: "View the full list of country codes" or "Contact us at stadia.dev/partnersupport"

Examples

Scenario

A user uploaded a game package, but it can't distribute to our cloud data centres because of some storage errors at the centres.

Ideally, these would never happen. But if they do, we want the partner to know the cause so they can act on it temporarily as they contact us.

Code	Original	UX
RESOURCE_EXHAUSTED	At least one of the requested locations does not has sufficient storage.	Can't distribute package. At least one zone doesn't have enough storage. This shouldn't happen. If you need to
		distribute your package right away, try archiving other packages to free up storage. Contact us at stadia.dev/partnersupport
NOT_FOUND	At least one of the locations has no capability to serve packages of this class.	Can't distribute package. At least one zone isn't configured to store your package. This shouldn't happen. Contact us at stadia.dev/partnersupport
(any other code)	Can't upload package due to an unknown error, but we're retrying in the background. If this error persists, try archiving the package and re-uploading. If it still persists, Contact Us for support.	Can't distribute package because of an unknown error. We'll keep trying to distribute the package. If this error persists, try archiving and unarchiving the package to restart the process. If this still persists, contact us at stadia.dev/partnersupport

Scenario

Various API messages are stored in a centralized location in the codebase. Some use out-of-date language.

I updated the messages to use our latest terminology, and cleaned them up while I was at it.

Original	UX
Invalid time range '%s'-'%s'. Start time must be less than end time.	Invalid time range. The start time must be before the end time.
Gamelet shape '%s' is invalid. Please choose a different gamelet shape and try again.	The instance shape '%s' is invalid.
Gamelet shape '%s' is incompatible with the pool's configuration. Please choose a different gamelet shape and try again.	The instance shape '%s' is incompatible with the pool's type or generation.
GCS URI in field '%s' is invalid.	The field '%s' has an invalid Google Cloud Storage URI. Make sure the URI begins with 'gs://'.
Enter a different release date as it cannot be in the past.	The release date can't be in the past.
Enter a different release date since the storefront cannot released on a US holiday.	The release date can't be on a US holiday.
Enter a different release date since the storefront cannot be released on a weekend.	The release date can't be on a weekend.
The release date you've selected is too close to today. Enter a release date at least 5 business days in the future.	The release date is too soon. It must be at least 5 business days in the future.

Austin Fisher

UI Content Strategy

Designing with words

UI Content Strategy

Austin Fisher

A big part of the UI work that I do is microcopy. You know, those buttons, table headings, instructions, and pesky errors.

But I also get deeply involved in the user flows, the design thinking, and the research. Often times, I'll tag-team with the UX Designer to map out the user journeys, discover gaps for opportunities, and even get my hands dirty in design tools. (Figma is my favorite of them all)

In this section you'll see:

- Case Study: Creating a promotion (End-to-end flow)
- UI Writing Samples
- Content Strategy + Style Guide excerpts

Austin Fisher

Case Study: Creating a promotion

UI copy & Information architecture

UI Content Strategy: Creating a promotion

Austin Fisher

Problem

Our partners needed to create and submit promotions to Stadia for their published games.

Solution

Create a web form within our Partner Portal to create, build, and submit a promotion on any number of products in a game.

My Role

I created new content strategy around web forms (this was our first one!) and worked close with UX Design and PMs on delivering a strong V1 experience to partners.

Things I did:

- Information architecture (content hierarchy)
- Content strategy
- UI text, micropcopy, and regular length copy.
- Error messages

UI Content Strategy: Creating a promotion

Austin Fisher

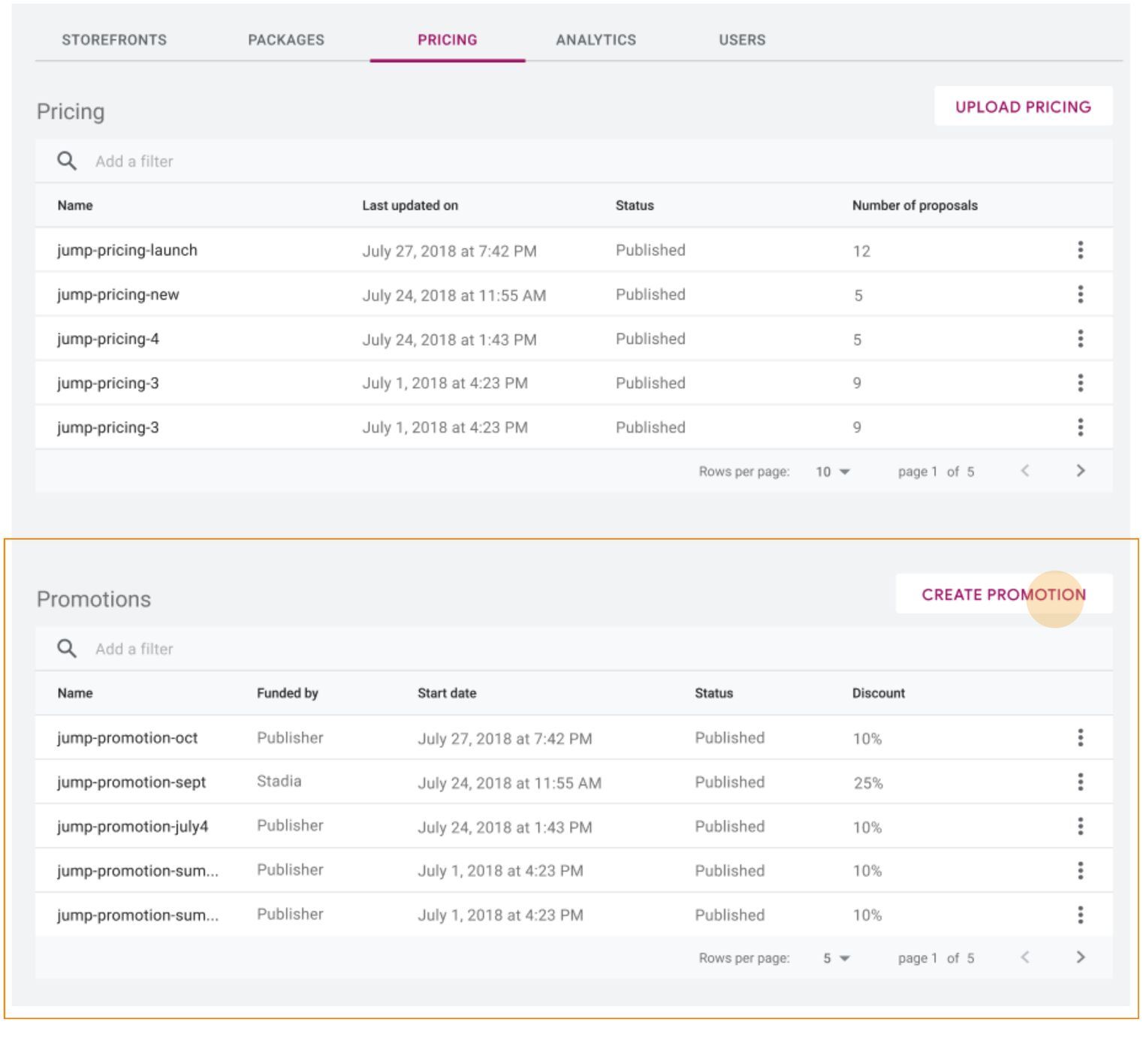
Step 1: The Promotion Dashboard

IA: Placing a new entity

Problem: Where do we put promotions?

My impact (Design thinking): A promotion is based off of published prices. You can't create a promotion until you've published prices. We already have a dedicated category for a game's prices.

Solution: We nested promotions within the "Pricing" section.



Microcopy: The right button

Problem: How do we start the promotion flow?

My impact: I had defined a content strategy guide to button naming, so drew from our existing content patterns to write this button.

Solution: Since this would launch the form and create a new entity in our backend systems, I chose "Create".

"Create" also established a good mental model. It alluded to an underlying process — "creating" isn't viewed as one step, but the result of many steps.

Strategy guide: Common buttons

Add	...an existing item to another item
Assign	...an existing item to a team
Create	**...an original item**
Upload	...a new file
New	...original item with 2+ above actions

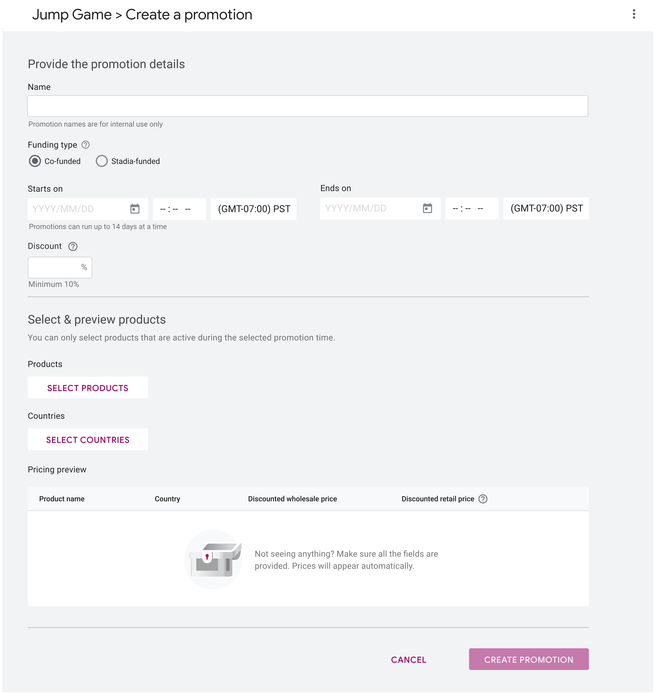
Step 2: Filling in the form

Content strategy: 2-step promotion form design

Problem: What information do we need? How do we ask for it?

My impact:
- Talked to PM to learn what questions we needed to ask, and what information Stadia required.
- Reviewed user research about common industry terms.
- Researched form design standards and principles.

Solution: I ensured we followed best practices for form design, and created form content principles:
- Only ask what we need to know, and don't make the user think too much.
- Make sure all restrictions and requirements are clearly labelled on each field, like formatting
- Split the form into sections that reflect when the user changes how they think.
- Let the fields speak for themselves. Only include more context when necessary.



Step 1: Set up the metadata, discount rate, and time restrictions of a promotion.
i.e. "The Basics"

Step 2: Choose the combination of products and countries to discount.

Note: At launch, only 1 discount per promotion was allowed.

Microcopy: 'Simple' vs 'Human' time zone picker

Problem: What's the best practice for choosing time zones? Simplicity, or human-readable labels?

My impact:
- Created the content pattern for time zones
- I talked with the PM about our requirements: Do we need time zones? Are we accounting for daylight saving time? What do we anticipate our partners choosing?
- I asked UX Design what the best strategies are for choosing from very long lists. Specifically, if we had to use dropdowns, or if we could use an input dropdown that searches.
- I asked engineering how complicated time zones needed to be, what logic we could build, and how big of a list we could store.
- I researched best practices for time zones across Google, and digital best practices. (UTC vs GMT)

Solution: Based on engineering's feedback and clean design principles, I created V1: a simple list of each possible time zone difference.
However, further research and partner feedback proved that people aren't usually confident with their time zone, so I made V2, a list of every local time zone. This was received more positively.

V1: Simple

V2: Human

IA: Splitting products and countries

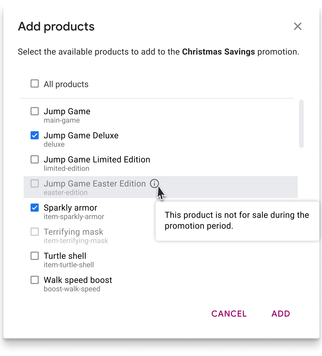
Problem: How do we increase efficiency and decrease human error when selecting products & countries?

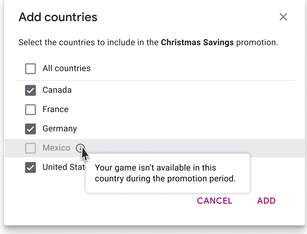
My impact:
- Worked with UXD to simplify the interaction into 2 tasks
- Discussed backend restrictions with engineering
- Reduced content to only show essential pieces of information
- Ensured we followed our dialog content standards

Solution: We implemented separate dialogs for products and countries. The preview automatically listed all the valid product/country combinations from the selections.

Pros: Broke down a big task into 2 simple tasks. Prioritized efficiency and automation. Reflected our backend system.

Cons: Restricted cusotmizability and didn't allow exceptions.





Strategy guide: Writing dialogs

Use consistent verbs	- Add [item] & "ADD" button - "Select" items & selection boxes
Provide instructions & context	Instructions: "Selecting..." Context: "the **Christmas Savings** promotion"
Explain exceptions	Any invalid products or countries have a tooltip explaining the issue.

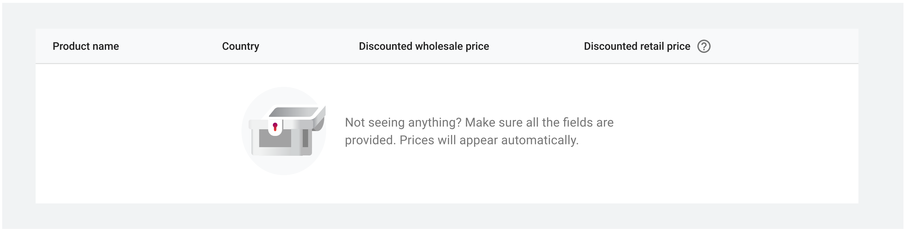
Step 3: Previewing and validation

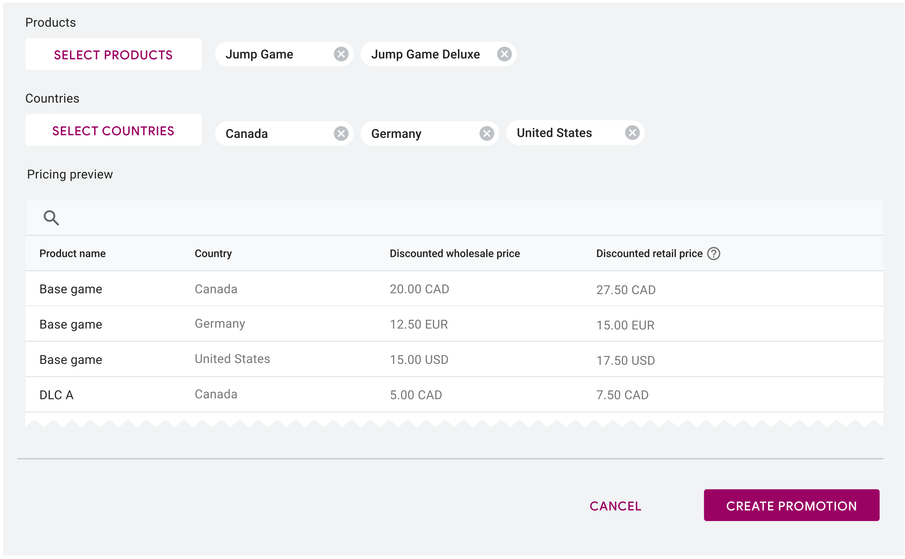
Microcopy: Empty states and table design

Problem: How should previewing work? What do partners want to preview?

My impact:
- Wrote a helpful empty state to explain how previewing worked
- Advocated to PM for plain language with legal phrases. “Indicative retail price” -> “Discounted retail price”
- Took out unneeded columns to make the important information easily scannable
- Implemented currency content strategy

Solution: The preview appeared automatically, once products and countries were chosen, presenting the most essential content (product/country pair & new prices) in a clear, easily scannable way.



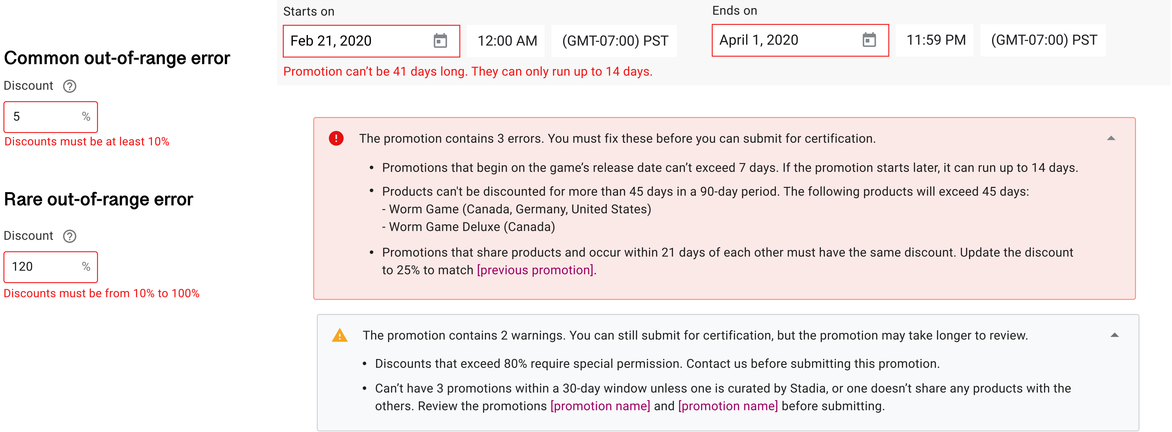


Error strategy

Problem: How do we communicate different types of errors so it still feels like a unified experience?

My impact:
- Implemented our error strategy to keep in-line errors short, and backend errors descriptive.
- Worked with PM to identify common errors and UXD to prevent users from making those errors with intelligent input fields
- Wrote errors closely with our internal Operations team to ensure the errors accurately represented our rules and restrictions
- Separated rare edge cases to improve the emotional response and avoid: “Obviously a discount can’t be more than 100%.”

Solution: We separated backend errors into message bars so they had space to be descriptive, and styled in-line errors around input boxes only. I ensured the errors used similar language to feel consistent.



Aftermath

I was very satisfied with how the promotion form turned out. The majority of partners were satisfied with the promotion form.

Since our UX exploration happened after product planning, some important functionality that we discovered got pushed back to V2 or even V3, because there just wasn't enough time.

Things we plan to include in V2:
- Multiple discount rates in one promotion
- The ability to include and exclude any product/country pairs.
- The ability to edit a single product/country pair.
- Stronger backend logic so we can shift most backend errors to in-line errors.

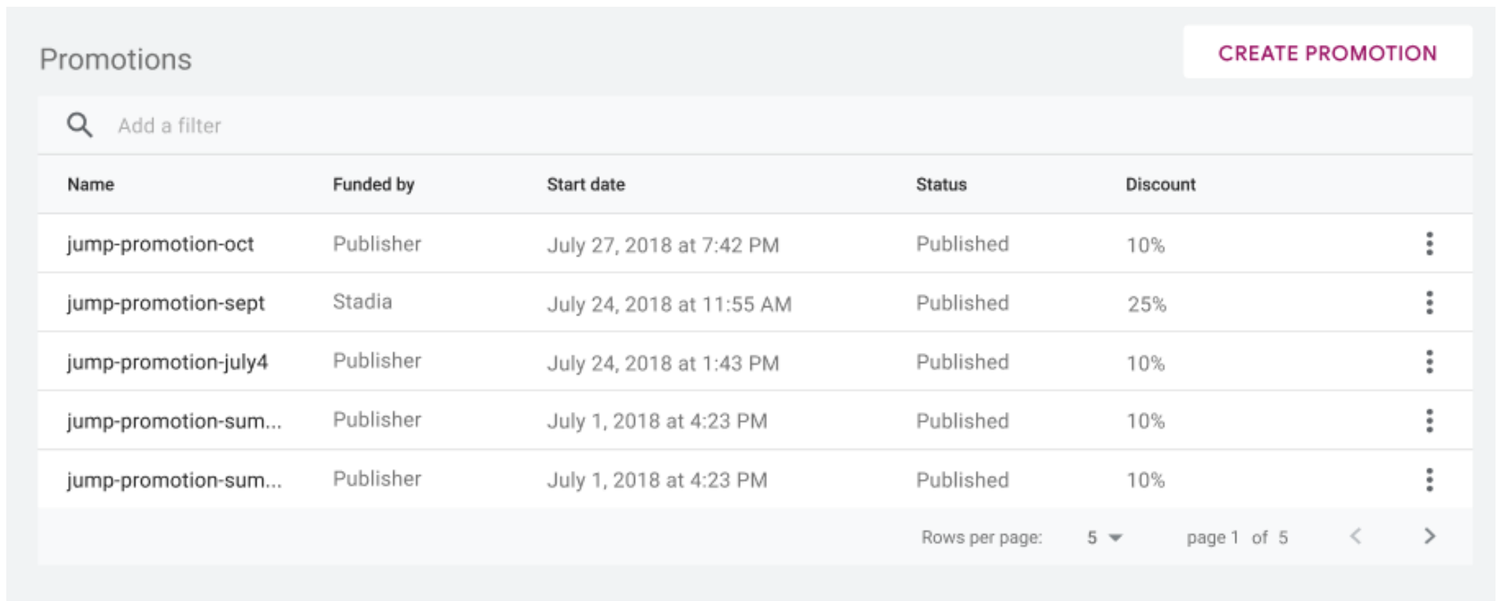
What I Would Change

"All promotions" table

I didn't get a change to provide input to the "All promotions" table (under the Pricing tab) during production. The form design was a higher priority at the time.

Problem: What's the most important thing to know about all my promotions as a publisher?

Solution: Group the columns into similar ideas that flow logically, add the end date, and modify our "date and time" content strategy in tables.



Rewrite: Group by priority

Add a filter

Name	Status	Discount	Funded by	Start date	End date
jump-promotion-oct	Active	10%	You	Jan 29, 2020	Jan 31, 2020
jump-promotion-sept	Ended	25%	Stadia	Jul 24, 2018	Jul 24, 2018
jump-promotion-july4	Ended	50%	You & Stadia	Jul 4, 2018	Jul 4, 2018
jump-promotion-sum...	Ended	10%	You	Jul 1, 2018	Jul 1, 2018
jump-promotion-sum...	Cancelled	10%	You	Jul 1, 2018	Jul 1, 2018

The name identifes the promotion, and the status tells partners how to interact with the promotion.

Since these are actionable items, they're up at the front where they can be scanned quickly.

"Discount" and "Funded by" are closely related ideas, creating a full 'thought' together.
"This discount is funded by..."

I also simplifed "Publisher" to "You", to make it feel more human and collaborative.

"Start date" is helpful, but is only half of the story. I added in "End date" to show the full lifecycle of a promotion.

I omitted the exact timestamp to keep the table scannable. They can still see the time if they click into the promotion.

Writing samples

Dialogs and my UI Style Guide

UI Content Strategy: Writing Samples

Austin Fisher

A developer setting up local hardware - final step

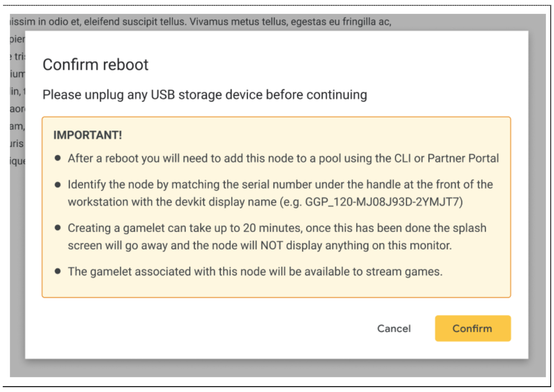
Problem: The hardware can't display information after it's rebooted due to technical limitations, but there are important instructions.

My impact: I took initiative to reorganize the information into more digestible steps, and added some friction to the process.
External to this flow, I also made sure our online documentation included these instructions.

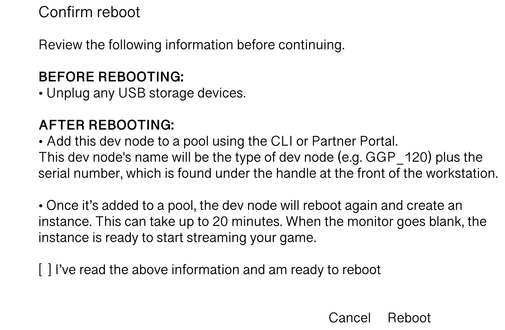
Solution: Clearly explain what to do before and after rebooting, and have the partner acknowledge that they've read the information.

Notes:
"Gamelet" was renamed to "instance" prior to this project.

OLD:



NEW:



Adding users to any team (developing project, publishing title, organization)

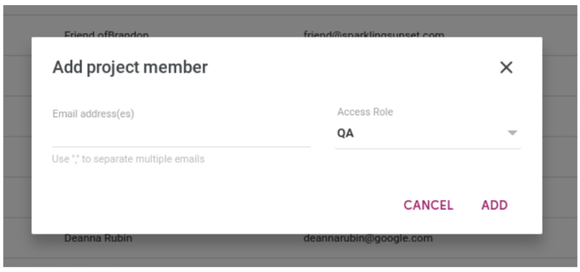
Problems:
1) Partners don't realize they can have more than 1 access role at a time.
2) It's unclear what the relationship is between "Users" and "Members" in the web portal.

My impact: I created a template for all team dialogs, so they all read the same way, simplifying our codebase and message.

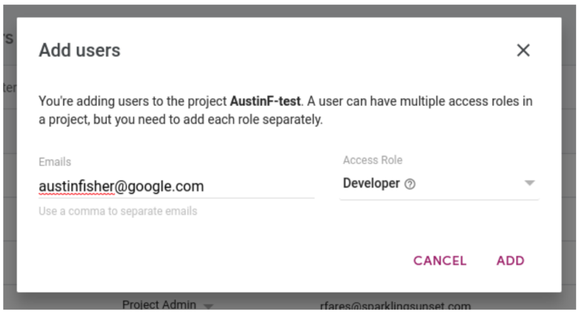
We used to say "project members, title members, & organization users" but in our system, they're all treated the same, so I eliminated the redundancy in favor of "users".

Solution:
Explain how access roles behave, and standardize the name "users" across all teams.

OLD:



WITH UPDATES:

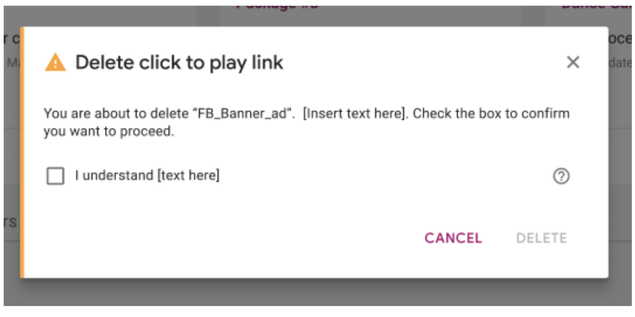


Deleting an active Click to Play link

Problem: Deleting a C2P link can negatively affect an ad campaign.

My impact: I contacted the ads team to learn exactly what happens with a link, so I could provide helpful context to partners.
I applied our dialog content patterns to ensure a consistent experience.

Solution: Warn partners of the consequences, and enforce an acknowledgement to stress the importance.



Delete a Click to Play link

You're deleting the **[name]** link. It will still appear in reports, but if you use the link in ad campaigns, it will direct players to the Stadia home page instead of your product **[product name]**. You can't undo this action.

[ ] I understand that this link will no longer direct players to **[product name]**

CANCEL DELETE

Dialogs

SUMMARY

A dialog is an on-screen prompt used to communicate additional information with a user. It can be used to confirm actions, or to gather information to complete actions.

There are **3** types of dialogs.

Confirmation dialog
This is used to slow the user down and see their action in context. It's used for destructive or significant actions, and provides a summary of what will happen next. It sometimes has a checkbox for confirmation.

Action dialog
This is used to gather more information to complete the action. Usually it contains instructions, and requires selecting from a list, or inputting fields. There are many variations of these, but they all follow the same structure.

Details dialog
This is used to display information for the user. There are no interactions other than copying or downloading information.

Does this action need a dialog?

If the action:

- Affects other people's work, is destructive, or is difficult to undo... → CONFIRMATION DIALOG
- Requires additional actions or more information to succeed... → ACTION DIALOG
- Surfaces view-only information... → DETAILS DIALOG

CONFIRMATION DIALOG

Contents

Heading
- Same verb as the button you pressed.
- Use sentence case.
- Specify the noun, but **do not** use variable names.

Contextual info
- Always in the style "You're doing this action to this item."
- Include all relevant pieces of data & variables, **bolded**.
- An alternate way is using a gray box (shown below)

Consequences / What this will do
- Explain what will happen:
  - Who this affects
  - How long it might take
  - Side effects / consequences
  - What to expect after
- (Optional) "You can't undo this action." if severe.

Checkbox (optional)
- If destructive, use to confirm the action
- Focus on the items being affected
- Must be checked to enable button

Buttons
- The first button is always "Cancel"
- The second button should match the verb in the heading of the dialog

Alternate Method:

Tips & Tricks

1) Don't put other actions in the dialog. It's only meant to confirm a decision.

2) To figure out if something's destructive, decide "can this immediately impact someone's work?" and "is it difficult to undo this action?" If yes, use a checkbox.

3) A checkbox may not be needed if a user performs the action regularly as part of their role. It would then be frustrating to check a box every time.

ACTION DIALOG

Contents

Heading
- Same verb as the button you pressed.
- Use sentence case.
- Specify the noun, but **do not** use variable names.

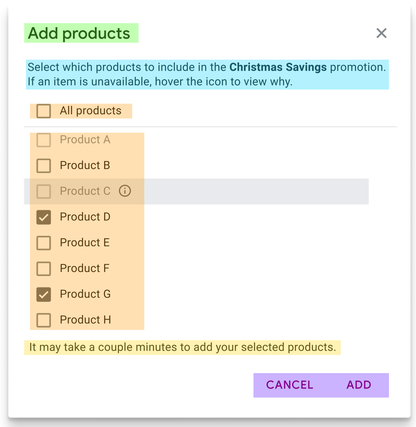
Contextual info
- Instructions of what to do in the dialog.
- Start with a verb the same as the actions taken within the dialog. "Select", "Fill in", "Specify".
- May provide a quick definition for context
- Contains relevant variables and names, to understand what's being affected, and how.

Actions
- Any actions go here:
  - drop-downs
  - selection boxes
  - upload buttons, etc.
- You may need more contextual info between certain actions, if the actions aren't related.

Consequences / What this will do (optional)
- Simple actions usually won't need this section
- Explain what will happen:
  - How long it might take
  - Side effects / consequences
  - What to expect after

Checkbox (optional)
- If destructive, use to confirm the action
- Focus on the items being affected
- Must be checked to enable button

Buttons
- The first button is always "Cancel".
- The second button should match the verb in the heading of the dialog



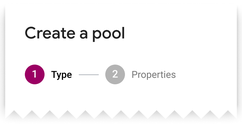
Tips & Tricks

1) If each action requires different instructions, split into a stepper.

- The action buttons would then be "CANCEL NEXT"
- Buttons on the second page would be "BACK ACTION"

The action would match the header. e.g. "CREATE".

DO



2) Instructions should be on the dialog, not hidden in a tooltip. If there's important information to know that could change someone's action, it must go on the dialog.

3) For common actions, shorter is better. "Adding a product" is a low-risk, common action for promotions, so we don't need to explain too much else.

Consequences are only needed if something unexpected may happen (waiting awhile, permissions change, etc.)

DETAILS DIALOG

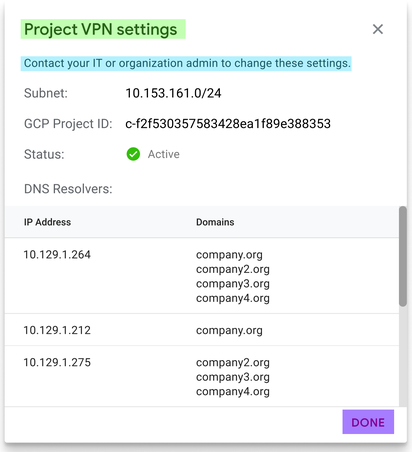
Contents

Heading
- Follows the format:
[Item] details / settings / configuration
- Use sentence case.
- Don't use a verb.

Contextual info
- Briefly explains what's on the page, or provide any instructions/directions related to the information. This is flexible.

Buttons
- Only one button, usually "Done".

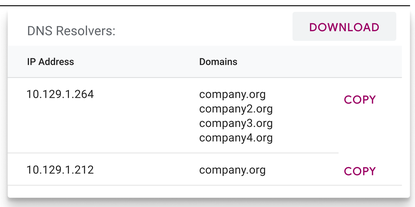
Other notes
- The details can have actions like "Copy" or "download". These shouldn't impact anything on the rest of the page.



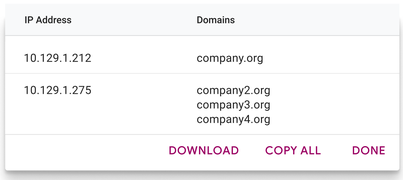
Tips & Tricks

1) Other actions are okay, but should be inline with the object thye're acting on (Copy text, download file).

DO



DO NOT



Buttons

Contents

CREATE STOREFRONT

Common buttons

The same word should be used for the same action across the portal. Here are words for common buttons.

CREATE [NOUN]	Create a new, original entity in our system with metadata. Used mostly for forms. e.g. “Create TSF”
ADD [NOUN]	Add an existing entity to another entity, or for adding people. e.g. “Add dev nodes”
ASSIGN [NOUN]	Assign an existing entity to a team (project/title). Not for people. e.g. “Assign application”
UPLOAD [NOUN]	Upload a new, original entity into our system. e.g. “Upload pricing”
NEW [NOUN]	Use when creating a new entity has multiple potential ways. e.g. “Upload” or “Transfer” a package.
SUBMIT [NOUN]	Submit an entity for certification.
DOWNLOAD [FORMAT]	Download a file in a certain format, if the format isn’t already clear. e.g. “Download CSV”
SAVE	Save progress on an entity. Not for downloading. Used mostly on forms.
EDIT	Edit the metadata of an existing entity. Paired with “Update” as a dialog action.

Tips & tricks

	DO	DO NOT
1) Keep it short and sweet. Don’t use **determiners** like “a” or “the”.	CREATE STOREFRONT	CREATE A STOREFRONT
2) Words should be impactful and straightforward.	DOWNLOAD EDIT	GET CHANGE
3) Re-use the same language from descriptions or headings.	Select the item you want to download. Select... DOWNLOAD	Select the item you want to download. Choose a file... SAVE

Austin Fisher

CLI Content Strategy

Style guide and writing samples

CLI Content Strategy

Austin Fisher

The Command Line Interface (CLI) is a text-based interface where partners can use the Stadia SDK to its full potential. This is an excerpt from my style guide, plus some samples below.

CLI style guide excerpt

How to write commands: Short and sweet

CLI commands are meant to be as brief as possible to minimize typing and maximize doing.
A command has 4 pieces: the group, the action, the item, and the flags.

```
ggp user add austinfisher@google.com --project="Test-Project1"
```

Group: A noun. A group is a what we're acting on. You can have nested groups, but these only make sense if the top group has too many commands or flag combinations.

Action: A verb. It should match what we use in the Partner Portal (Add, Assign, Create, Upload), whenever possible.

Item (optional): The ID or name of an entity in the group. You can have multiple items.

Flags (optional): Any other variables required for the command to work. Boolean flags start with a verb like 'enable' and always have a 'true' state. Item flags just use the name of the item.

How to write help text: Long and descriptive

This is very similar to technical writing. Help text explains what a command can do in complete detail, covering every use case.

Group help: Provides a definition of the group. It almost always begins with "Manage", and explains the user benefit of the item.

Action help: The action help has 4 pieces.
1) The core functionality.
2) How to use all the required flags or arguments, providing context that may affect user input.
3) How to use the optional flags, how they work together, and their user benefit.
4) Any other helpful information, other commands to use first, documentation to read, etc.

Flag help: This describes how to format the flag. It lists the arguments and formats it accepts. These don't require full sentences.

CLI writing samples

Connecting a metrics tracker before a session starts

Problem: Users can't connect the metrics tracker until after the game begins.

Solution: Create a command to connect and disconnect the metrics tracker.

My impact: Looking at the user journeys, I added a "disconnect" command since it's possible to connect to the wrong instance.

CLI - Original

Original command	Original short	Original long	Original flags
ggp chrome-extension connect	Connect to the game on your reserved instance	Connect to the game on your reserved instance	

CLI - UXW wording

UX command	UX short	UX long	UX flags
ggp chrome-extension	Manage the remote chrome extension	Manage the remote chrome extension. Connecting this extension to a reserved instance provides metrics and run-time stats about the instance and the stream. It provides this data regardless of which screen you're playing on, and displays the data on your workstation's screen.	
... connect	Connect the remote chrome extension to a reserved instance	Connect the remote chrome extension to a reserved instance. This defaults to your current instance. You can specify a different one with the --instance flag.	--instance: Instance ID
... disconnect	Disconnect the remote chrome extension from a reserved instance	Disconnect the remote chrome extension from a reserved instance. This defaults to your current instance. You can specify a different one with the --instance flag.	--instance: Instance ID

Setting up a crash report bucket & downloading reports

Problem: Users needed to download crash reports.

Solution: Create commands to create a crash report bucket and to download the reports.

My impact: I renamed a proposed command from '**set-gcs-destination**' to 'gcs-bucket **update**' to follow content standards.
This nested group also made it easy to separate the bucket management from the actual reports, especially since more commands would be coming to GCS buckets soon.

I also talked with the engineer to merge 'count' into the 'list' command to simplify how to view information.

UX command	UX short	UX long	UX flags
ggp crash-report	Manage crash reports	Manage crash reports. You can set a Google Cloud Storage (GCS) bucket to store crash reports, and then download the stored reports.	
ggp crash-report gcs-bucket	Manage the GCS bucket destination for storing crash reports	Manage the Google Cloud Storage (GCS) bucket destination for storing crash reports	
... update	Update the GCS bucket for storing crash reports	Update the Google Cloud Storage (GCS) bucket for storing crash reports. Specify the GCS bucket URI with the --bucket flag, and specify which title's crash reports to store in this bucket with the --title flag.	--title: Title ID --bucket: GCS bucket URI. Example: gs://bucketname/optional/path/to/folder
ggp crash-report report	Manage and view crash reports	Manage and view crash reports. You can list and download crash reports, and can filter results by different times or specific report IDs.	--title: ***(same as above)***
... list	List crash reports and details	List crash reports and details. Specify which title's crash reports to list with the --title flag. It defaults to listing all reports created over the past 24 hours, but you can change this with the --start-time and --end-time flags. You can also list specific reports with the --report-id flag.	--title: ***(same as above)*** --start-time: Default is 24 hours ago. Use RFC 3339 format: YYYY-MM-DDThh:mm:ss(+/-)(UTC offset) --end-time: Default is now. Use RFC 3339 format: YYYY-MM-DDThh:mm:ss(+/-)(UTC offset) --report-id: Crash report ID
... download <report-id>	Download crash reports	Download crash reports. Specify which title's crash reports to download with the --title flag, or specify a report ID. Set the save destination with the --path flag. It defaults to downloading all reports created over the past 24 hours, but you can change this with the --start-time and --end-time flags, or by specifying a report.	--title: ***(same as above)*** --start-time: ***(same as above)*** --end-time: ***(same as above)*** --path: Local path to save the report
... describe <report-id>	Describe a crash report		

Austin Fisher

Naming & Ideation

Solving a “word problem” in 24 hours

Creating a name: “Betas” to “Custom Audiences”

What’s in a name? Well, as it turns out, a lot. A name carries a lot of meaning, emotions, and reactions. Where “Project Stream” was a codename that invoked curiosity and mystery around streaming innovation, “Stadia” was a metaphor for all things play, inspiring community, action, and excitement.

Summary: “What do we call betas?”

Problem

I was approached to help name “betas”. However this name couldn’t be “beta”, and it needed to be applied to many different items. Those items needed renaming too. What do we call this group of unfinished and fragmented games? What actually needed a name?

My approach

1) I challenged the idea of what a “beta” was, and did a quick exercise to group similar items.
2) I researched industry terminology to see what words both partners and players were already using.
3) I questioned the premise for not using “beta”, and persuaded the team to consider changing our legal documents instead, so we use the right word everywhere.
4) I brainstormed with the product manager some alternatives and documented our ideas.
5) I suggested a more plain-language, clear alternative to the PMs proposed final name, and got immediate cross-functional approval.

Step 1: Categorizing the items

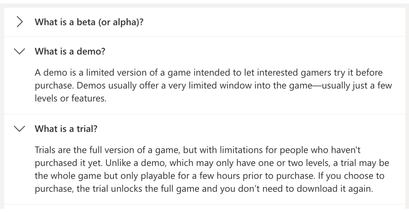
I noticed in the proposal that there were 2 distinct categories of items. There were unfinished games for testing purposes, and finished games for generating sales. I organized them below:

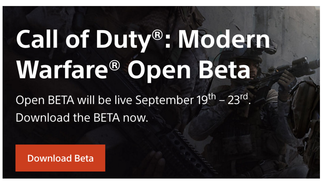
Unfinished Game (Testing)	Finished Game (Sales)
Open Betas	Free Trials
Closed Betas	Demos
Early Access Testing	

These were all very common terms. I suggested we use the same terms internally and externally. But since “beta” was already used as a different publishing milestone in our legal docs, that had to change.

Step 2: Industry research

The question I kept hearing was “What do we call the category that contains betas?” I struggled with the question. Did the category even need a name? Could we simply say “Game version” or “Game type”? So I looked into the words the industry used.





I checked the stores of other game platforms like Nintendo, PlayStation, and Steam. I also checked websites for game developers and publishers to see what language they used.

Across the board, everyone used **Open Beta** and **Closed Beta**. The category was either called **Early Access** or there was no category name at all. Demos and free trials were all called **demos** and **free trials**. I sourced the various websites when bringing my findings to the PM.

Step 3: Questioning the premise

I asked why we wanted to change the name from **beta** to something else, when **beta** was the common industry term.

There was a legal reason. We identified certain publishing milestones as **Alpha** and **Beta**, and these would come much earlier than an **Open** or **Closed Beta**. This would make partner deliverables confusing.

However... this didn’t sit right with me. We should always favor the most accurate term, even if it means extra work. The PM agreed with me, and we proposed to change the legal milestone term instead.

Step 4: Brainstorming

We still had a problem: “What do we call the category that contains Betas and all other game versions?” We cycled through a few options:
- Game versions
- Releases
- Release versions
- Limited audience releases

I proposed “game versions” to start. These words came most naturally when talking about the concept. But the PM wanted something that highlighted the publisher value, and proposed “Limited audience releases”.

But “Limited” didn’t seem right. It was too narrow to describe early access programs that could have millions of players. What was limited? The time window? The audience?

Step 5: The final name! Using common language ”Betas” & “Custom audiences”

I asked what the main benefit was. “Choice” was the word that came out most. “Chosen audiences”? “Selected audiences”? The next day, the PM sent an email out about the new name: “Settable audiences”.

Red flags went off in my head. Is **settable** a common word? Is it a word people ever use? I feared it would still create confusion.

So I searched for similar words, opened the thesaurus, and thought to myself “what’s great about a settable audience?” “What actionable value does it produce?”

The answer: “You can customize who can play this version of the game.” And there it was. Our name. “Custom audiences.”

I suggested “Custom audiences” to all the cross-functional stakeholders, and got immediate positive feedback. Everyone was on board.

My impact

I took a big concept, identified the root cause of the problem, proposed meaningful, well-researched options, and used my writing knowledge to find common, plain-language words that conveyed exactly what the product manager intended.

In 24 hours, I took a vague idea and turned it into a confident, clear, and actionable final name.

Thanks for reading!

Come talk to me about UX and poutine.

linkedin.com/in/stuxain/