Error Messaging Guidance

Error alert fundamentals

Error alerts are a subcategory of alerts used to inform that something went wrong and attention is needed. Error in-app alerts provide actionable messaging in context, and they're always red.

Error messages typically meet our customers during moments of frustration. An error itself is not helpful but error alerts can be helpful by bringing clarity to a frustrating situation.

Always consider and discuss with your team how you can prevent an error before it happens. Consider finding ways to steer the user away from potential errors with tool tips and instructions. Remember, a well-written error alert for an error that shouldn't exist in the first place gives an overall negative user experience.

Content

An error alert is a notification that something has gone wrong. Notifications typically follow the pattern of explaining **what happened**, **why it happened** and **what you can do about it**.

However, because errors are a negative experience, there is an urgency to resolve the issue which changes the content pattern slightly. The ideal error alert should anchor the error in relevant context about what went wrong and give a path to resolution as efficiently as possible.

Prioritize **what they can do about** the error

For some instances, what went wrong and why it went wrong aren’t relevant. This could be because it is a user error. Our technology is functioning like it should and we certainly don’t want to point blame at the user. The message would only need to tell the user what they should do to resolve the error. Example: Please enter a valid email address.

Avoid over-using “try again” or “check back later”

With many errors, the only thing the user can do is either immediately try again or to check back later when we’ve hopefully fixed the error on our end. Use the phrase, “Please check back later.”

However, don't suggest an action that will not be helpful to the user. Waiting and/or reloading is a common solution but it is not the only solution. Always discuss with your team whether retrying will really change anything and if there is a better action that the user can take.

You can also turn this error into a positive opportunity by highlighting why the user would want to check back later (ex: Check back later for important messages and helpful tips related to your account.)

Explain **what happened** so they don't feel lost

In other instances, it is necessary to give some context of what went wrong. Don't settle for a generic "whoops" or "snag" message when you can provide a specific reason or the most plausible one. The goal is to find the right balance of general/specific: general enough to avoid technical jargon, but specific enough that customers generally understand what happened.

For example, when a feature fails to load, the user has no way to know what should have happened. They may assume that a broken experience is functioning correctly unless we briefly explain what went wrong.

Giving more context on what happened is best done as a statement that doesn't avoid blame with passive voice or shift blame on an inanimate app. Instead, it begins with "We're having trouble…"

In some instances, explaining what went wrong is enough to imply what the user should do about it. In these cases, don't restate the error as a command just to conform to a content pattern.

> Don't: This email address is already in our system. Please enter an email address that isn't in our system.
>
> Do: This email address is already in our system.

Include **why it happened** to help them avoid repeated errors

Typically, users may or may not care to know about APIs and the difference between retrieving and displaying data. This means the why the error happened is usually only relevant if it has an impact on what they would do to resolve the error.

For example, with policy errors, explaining why the error happened helps the user avoid making the same mistake twice. This is best done as a positive statement that focuses on the policy, not the violation. Remember, the user only cares about what they can do, not what they can't do. If and only if it's necessary, you can link out to additional policy guidance.

As with explaining what went wrong, if explaining the policy if enough info for the user to understand what to do next, don't restate the policy as an action to conform to a content pattern.

> Don't: You need at least $11 in this account to order 100 or more checks. Add at least $11 to this account.
>
> Do: We accept images only in JPEG or PNG format. Please upload your image in the correct format.

Empathize without using apologies

Research shows that a blanket apology statement like "We're sorry" doesn't sound as empathetic as we may think it does. The best way to empathize with the user is to emotionally understand the urgency of resolving the error.

If the error is more on our end than theirs and there isn't a specific, immediate action that you can recommend to resolve the error, use "...but we're working on it" as an alternative to apologizing.

However, don't include this if you are writing for an error where it's unlikely that anyone is working on the error.

Avoid technical language

Loading is preferred to retrieving because the user can't see behind the scenes and failing to find something is more scary than failing to display something. Likewise, showing is preferred to loading.

Typical content patterns

There are four broad types of errors:

1. Data/feature unavailable - when an experience or part of an experience fails to load because the data doesn't exist, the system can't retrieve the data, the feature is down for maintenance or the customer's device fails to load the feature.
2. Input-level error - when a customer enters the wrong or invalid info into a form field.
3. Policy violation error - when a customer attempts to take an action that is prevented by a Capital One policy (even if it's a benign policy like only allowing photo uploads in a specific format, which is a far cry from being criminal).
4. Fatal error - when access to all of EASE is walled off and waiting on an offline customer action.

A more thorough explanation of error types can be found on this confluence page.

After determining your error type, use the table below to find a pattern that fits your use case.

Error Messaging Content Pattern						
Data/Feature Unavailable				Policy Violation	Fatal	Input-level Error
Generic	Result-focused	Action-focused	Feature-focused			
Something went wrong	Something went wrong	[User's next best step]	Something went wrong			
We're having trouble showing that info, but we're working on it.	We're having trouble [completing x action] right now, but we're working on it. [Describe the result].	We're having trouble [completing x action] right now, but we're working on it. [Describe the user's next best step].	We're having trouble showing your [feature] right now, but we're working on it. Check back later for [what the feature offers].	[Statement about the policy that focuses on the policy, not the violation]. [Action that follows policy, "Please try again" or skip this sentence].	We can't [action customer was expecting] until [customer/we take an action]. [How to take that action].	[User action to conform to form restrictions].
OK	OK	[Take step]				

Design

As a subcategory of alerts, error alerts follow gravity guidelines...

There are 3 types of components that can be used for an error alert:

1. Inline text and icon
2. Modal
3. Global alert bar

Inline text and icon

This treatment is preferred to modals and the global alert bar because it keeps the error alert close to where the error took place. The goal is to keep the red text of the error as contextual as possible. If a feature loads but the data within the feature isn't loading, the inline alert should take the space where the data would otherwise be.

Because inline text is contextual, use "this" when referring to the feature that failed. Example: We're having trouble showing you this info right now. We're working on it. Please try again later.

Modal

Modals interrupt the user experience and are necessary when...

Example use cases:

- when a user completes an action, but an error prevents the action to be truly executed
- when the entire experience fails when navigating between pages and is unavailable to the user
- when a feature is unavailable upon user interaction.

Because modals are slightly detached from the component or feature that failed, use "that" when referred to what failed. Example: We're having trouble showing you that info right now. We're working on it. Please try again later.

Global alert bar

If a user returns to a page with flagged inputted information...?

If using a flagging error, match the two error messages so the user doesn't think that they have two different errors.