Write and Rewrite with built-in AI

Apr 11, 2025 Mike Wasserman

Related material:

- Tsuyoshi Horo 's Using ai.writer/ai.rewriter API on Chrome
- Thomas Steiner 's developer.chrome.com/docs/ai/summarizer-api
- Thomas Steiner 's Built-in AI Early Preview Program - The Writer and Rewriter APIs - Update #5

Explainer

https://github.com/webmachinelearning/writing-assistance-apis

Chrome Status

Writer API - Chrome Platform Status
Rewriter API - Chrome Platform Status

Intent

Intent to Prototype: Rewriter API
Intent to Prototype: Writer API

Intro

Imagine you could offer your users the ability to start drafting a message, product review, or even formal documents. Additionally, you could offer them refinements to adjust the tone, length, and focus of their written drafts. Beyond that, you could generate written content customized for your users as they engage with your site.

The Write and Rewrite APIs can be used to generate different types of writing in varied lengths and formats, such as sentences, paragraphs, bullet point lists, and more. We believe these APIs are useful in the following scenarios:

Writer API

- Generating textual explanations of structured data (e.g. poll results over time, bug counts by product, …)
- Expanding pro/con lists into full reviews.
- Generating author biographies based on background information (e.g., from a CV or previous-works list).
- Break through writer's block and make creating blog articles less intimidating by generating a first draft based on stream-of-thought or bullet point inputs.
- Composing a post about a product for sharing on social media, based on either the user's review or the general product description.

Rewriter API

- Removing redundancies or less-important information in order to fit into a word limit.
- Increasing or lowering the formality of a message to suit the intended audience.

- Suggest rephrasings of reviews or posts to be more constructive, when they're found to be using toxic language.
- Rephrasing a post or article to use simpler words and concepts ("explain like I'm 5").

TODO: Get started (with Origin Trial integration)

Use the Writer and Rewriter APIs

First, run feature detection to see if the browser supports the Writer and Rewriter APIs.

```
JavaScript
if ('Writer' in self && 'Rewriter' in self) {
  // The Writer and Rewriter APIs are supported.
}
```

Model download

The Writer and Rewriter APIs use powerful AI models trained to generate high-quality writing. While the APIs are built into Chrome, the models are downloaded separately the first time a website uses the APIs.
To determine if the model is ready to use, call the asynchronous Writer.availability() and Rewriter.availability() functions. The method will return a promise that fulfills with one of the following availability values:

- "unavailable" means that the implementation does not support the requested options.
- "downloadable" means that the implementation supports the requested options, but it will have to download something (e.g. a machine learning model or fine-tuning) before it can do anything.
- "downloading" means that the implementation supports the requested options, but it will have to finish an ongoing download before it can do anything.
- "available" means that the implementation supports the requested options without requiring any new downloads.

To trigger the model download and create the writer or rewriter, call the asynchronous Writer.create() or Reriter.create() function. If the response to availability() was downloadable or downloading, it's best practice to listen for download progress. This way, you can inform the user in case the download takes time.

```
JavaScript
const writer = await Writer.create({
  monitor(m) {
    m.addEventListener('downloadprogress', (e) => {
      console.log(`Downloaded ${e.loaded * 100}%`);
    });
  }
});
```

API functions

The create() functions lets you configure a new writer or rewriter object to your needs. They each take an optional options object with the following parameters:

Common options for both APIs

- sharedContext: Additional shared context that can help shape the writing.
- expectedInputLanguages: The languages that will be used for input.
- expectedContextLanguages: The languages that will be used for context.
- outputLanguage: The language that should be used for output.

Additional Writer API options:

- tone: The tone to be used for writing, with the allowed values "formal", "neutral" (default), "casual"
- format: The format to be used for writing, with the allowed values "plain-text" (default), "markdown"
- length: The length to be used for writing, with the allowed values "short", "medium" (default), "long"

Additional Rewriter API options:

- tone: The tone to be used for rewriting, with the allowed values "as-is" (default), "more-formal", "more-casual"
- format: The format to be used for rewriting, with the allowed values "as-is" (default), "plain-text", "markdown"
- length: The length to be used for rewriting, with the allowed values "as-is" (default), "shorter", "longer"

Note: Once set, the parameters can't be changed. Create a new object if you need to make modifications to the parameters.

The following example demonstrates how to initialize the writer. The steps to initialize a rewriter are identical, except for the per-API options listed above.

JavaScript

```
const options = {
  tone: 'formal',
  monitor: (m) => {
    m.addEventListener('downloadprogress', e => {
      console.log(`Downloaded ${e.loaded * 100}%`);
    });
  }
};
const availability = await Writer.availability(options);
if (availability == 'unavailable') {
  // The Writer API isn't usable.
  return;
}

if (availability !== 'available') {
  // The Writer API can be used after the model is downloaded.
  console.log('Please wait while we load the model');
}

const writer = await self.Writer.create(options);
```

Run the writer or rewriter

There are two ways to run each API: streaming and non-streaming.

Non-streaming

With non-streaming, the model processes the input as a whole and then produces the output.
To get a non-streaming response, call the asynchronous write() or rewrite() function. The first argument for write() is the writing instructions, while the first argument for rewrite() is the text you want to rewrite. The second, optional argument for both is an object with a context field. This field lets you add background details that might improve the result.

```
JavaScript
const writerResult = await writer.write('A draft for an inquiry to my bank about how to enable wire transfers on my account', { context: 'I am a new customer'}
);

const rewriterResult = await rewriter.rewrite(reviewTextbox.textContent, {
  context: "Avoid any toxic language and be as constructive as possible."
});
```

Tip: When providing writing instructions or text to be rewritten, it's best to remove any unnecessary data such as HTML markup. For content present on the page, you can achieve this by using the innerText property of an HTML element with the targeted text, as this property represents only the rendered text content of an element and its descendants.

Streaming

Streaming offers results as they become available, rather than waiting for the result to be complete.
To get a streaming response, call the writeStreaming() or rewriteStreaming() function. Then iterate over the available segments of text in the stream.

```
JavaScript
const stream = writer.writeStreaming('A draft for an inquiry to my bank about how to enable wire transfers on my account');

for await (const chunk of stream) {
  composeTextbox.append(chunk);
}
```

These functions each return a promise that resolves to a ReadableStream, where the segments are successive pieces of a single long stream, not a concatenation of all text generated thus far.

TODO: Setting an AbortSignal

Demo

You can try the Writer and Rewriter APIs in the Writer / Rewriter API Playground.
Additional demos are available at the Built-in AI Playground.

Standardization effort

We're working to standardize the Writer and Rewriter APIs, to ensure cross-browser compatibility.

Our API proposal received community support and has moved to the W3C Web Incubator Community Group for further discussion. The Chrome team requested feedback from the W3C Technical Architecture Group, and asked Mozilla and WebKit for their standards positions.

Participate and share feedback

Enable developer testing before the origin trials are available by enabling these chrome://flags :

- chrome://flags/#optimization-guide-on-device-model (Enabled BypassPerfRequirement)
- chrome://flags/#writer-api-for-gemini-nano (Enabled)
- chrome://flags/#rewriter-api-for-gemini-nano (Enabled)

You can alternatively enable developer testing of these APIs with the following command line flag:

- --enable-features=OptimizationGuideOnDeviceModel,EnableAIWriterAPI,EnableAIRewriterAPI

Start testing the Writer and Rewriter APIs now by joining the origin trial and share your feedback. Your input can directly impact how we build and implement future versions of this API, and all built-in AI APIs.

- For feedback on Chrome's implementation, see existing bug reports, file a bug report or a feature request.
- Discuss the API design on GitHub by commenting on an existing Issue or open a new one.
- Participate in the standards effort by joining the Web Incubator Community Group.

Feature limitations

- Only English input and output is supported.