W3C Recommendation

Web Share API

W3C Recommendation 30 May 2023

▼ More details about this document

This version:
https://www.w3.org/TR/2023/REC-web-share-20230530/

Latest published version:
https://www.w3.org/TR/web-share/

Latest editor's draft:
https://w3c.github.io/web-share/

History:
https://www.w3.org/standards/history/web-share
Commit history

Test suite:
https://wpt.live/web-share/

Implementation report:
https://w3c.github.io/web-share/imp-report/

Editors:
Matt Giuca (Google Inc.)
Eric Willigers (Google Inc.)
Marcos Cáceres (Apple Inc.)

Feedback:
GitHub w3c/web-share (pull requests, new issue, open issues)

Errata:
Errata exists.

Browser support:
caniuse.com

See also **translations**.

Abstract

This specification defines an API for sharing text, links and other content to an arbitrary destination of the user's choice.

The available share targets are not specified here; they are provided by the user agent. They could, for example, be apps, websites or contacts.

Status of This Document

This section describes the status of this document at the time of its publication. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at https://www.w3.org/TR/.

This document was published by the Web Applications Working Group as a Recommendation using the Recommendation track.

W3C recommends the wide deployment of this specification as a standard for the Web.

A W3C Recommendation is a specification that, after extensive consensus-building, is endorsed by W3C and its Members, and has commitments from Working Group members to royalty-free licensing for implementations. Future updates to this Recommendation may incorporate new features.

This document was produced by a group operating under the W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

This document is governed by the 2 November 2021 W3C Process Document.

§ Implementation status

This section is non-normative.

The ability to share content is often dependent on the underlying operating system providing a "share" capability and also on OS UI conventions. For example, some OSs present a "share sheet", while others rely on an pop-up menu. Due to these aforementioned dependencies, there is ongoing work by implementers to bring the Web Share API to all OSs. This ongoing effort is reflected as failures in the implementation report, which is generated by running tests on a limited set of OSs. However, the Working Group is optimistic that the Web Share API will become more available across all OSs over time, and is already widely available on popular OSs across a range of devices.

Table of Contents

§ 1. Usage Examples

This section is non-normative.

§ 1.1 Sharing text and links

This example shows a basic share operation. In response to a button click, this JavaScript code shares the current page's URL.

EXAMPLE 1: Sharing text and URL

```
shareButton.addEventListener("click", async () => {
  try {
    await navigator.share({ title: "Example Page", url: "" });
    console.log("Data was shared successfully");
  } catch (err) {
    console.error("Share failed:", err.message);
  }
});
```

Note that a `url` of `''` refers to the current page URL, just as it would in a link. Any other absolute or relative URL can also be used.

In response to this call to `share()`, the user agent would display a picker or chooser dialog, allowing the user to select a target to share this title and the page URL to.

§ 1.2 Sharing a file

This example shows how to share a file. Note that the `files` member is an array, allowing for multiple files to be shared.

EXAMPLE 2: Sharing a file

```
shareButton.addEventListener("click", async () => {
  const file = new File(data, "some.png", { type: "image/png" });
  try {
    await navigator.share({
      title: "Example File",
      files: [file]
    });
  } catch (err) {
    console.error("Share failed:", err.message);
  }
});
```

§ 1.3 Validating a share

Calling `canShare()` method with a `ShareData` dictionary validates the shared data. Unlike `share()`, it can be called without transient activation.

EXAMPLE 3

```
const file = new File([], "some.png", { type: "image/png" });

// Check if files are supported
if (navigates.canShare({files: [file]})) {
  // Sharing a png file would probably be ok...
}

// Check if a URL is ok to share...
if (navigates.canShare({ url: someURL })) {
  // The URL is valid and can probably be shared...
}
```

§ 1.4 Checking if members are supported

Because of how WebIDL dictionaries work, members passed to `share()` that are unknown to the user agent are ignored. This can be a problem when sharing multiple members, but the user agent doesn't support sharing one of those members. To be sure that every member being passed is supported by the user agent, you can pass them to `canShare()` individually to check if they are supported.

EXAMPLE 4: Future-proofing shares

```
const data = {
  title: "Example Page",
  url: "https://example.com",
  text: "This is a text to share",
  someFutureThing: "some future thing",
};
const allSupported = Object.entries(data).every(([key, value]) => {
  return navigator.canShare({ [key]: value });
});
if (allSupported) {
  await navigator.share(data);
}
```

Alternatively, you can adjust application's UI to not show UI components for unsupported members.

EXAMPLE 5: Filtering out unsupported members

```
const data = {
  title: "Example Page",
  url: "https://example.com",
  text: "This is a text to share",
  someFutureThing: "some future thing",
};

// Things that are not supported...
const unsupported = Object.entries(data).filter(([key, value]) => {
  return !navigator.canShare({ [key]: value });
});
```

§ 1.5 Enabling the API in third-party contexts

The default allowlist of 'self' makes Web Share API available by default only in first-party contexts.

Third-party can be allowed to use this API via an `iframe`'s `allow` attribute:

EXAMPLE 6: Enabling the web-share API in a third-party context

```
<iframe
  src="https://third-party.example"
  allow="web-share">
</iframe>
```

Alternatively, the API can be disabled in a first-party context by specifying an HTTP response header:

EXAMPLE 7: Disabling Web Share over API HTTP using Permissions Policy

```
Permissions-Policy: web-share=()
```

See the *Permissions Policy* specification for more details and for how to control the permission policies on a per-origin basis.

§ 2. API definition

§ 2.1 Extensions to the `Navigator` interface

WebIDL

```
partial interface Navigator {
  [SecureContext] Promise<undefined> share(optional ShareData data =
{});
  [SecureContext] boolean canShare(optional ShareData data = {});
};
```

§ 2.1.1 Internal Slots

This API adds the following internal slot to the `Navigator` interface.

`Promise`? *[[sharePromise]]*
The this.`[[sharePromise]]` is a promise that represents a user's current intent to share some data with a share target. It is initialized to `null`.

§ 2.1.2 `share()` method

▶ MDN ✅

When the `share()` method is called with argument *data*, run the listed steps listed below while taking into consideration the following security implications.

Web Share enables data to be sent from websites to a share target, which can be a native applications. While this ability is not unique to Web Share, it does come with a number of potential security risks that can vary in severity (depending on the underlying platform).

The data passed to `share()` might be used to exploit buffer overflow or other remote code execution vulnerabilities in the share target that receive shares. There is no general way to guard against this, but implementors will want to be aware that it is a possibility (particularly when sharing files).

Share targets that dereference a shared URL and forward that information on might inadvertently forward information that might be otherwise confidential. This can lead to unexpected information leakage if shares reference content that is only accessible by that application, the host on which it runs, or its network location.

Malicious sites might exploit share targets that leak information by providing URLs that ultimately resolve to local resources, including, but not limited to, "file:" URLs or local services that might otherwise be inaccessible. Even though this API limits shared URLS to a restricted set of sharable schemes, use of redirects to other URLs or tweaks to DNS records for hosts in those URLs might be used to cause applications to acquire content.

To avoid being used in these attacks, share targets can consume the URL, retrieve the content, and process that information without sharing it. For instance, a photo editing application might retrieve an image that is "shared" with it. A share target can also share the URL without fetching any of the referenced content.

Share targets that fetch content for the purposes of offering a preview or for sharing content risk information leakage. Content that is previewed and authorized by a user might be safe to forward, however it is not always possible for a person to identify when information should be confidential, so forwarding any content presents a risk. In particular, the `title` might be used by an attacker to trick a user into misinterpreting the nature of the content (see also 5. Accessibility considerations).

As with any user of `DOMException`, implementors need to carefully consider what information is revealed in the error message when `share()` is rejected. Even distinguishing between the case where no share targets are available and user cancellation could reveal information about which share targets are installed on the user's device.

1. Let *global* be this's relevant global object.
2. Let *document* be *global*'s associated `Document`.
3. If *document* is not fully active, return a promise rejected with an "`InvalidStateError`" `DOMException`.
4. If *document* is not allowed to use `"web-share"`, return a promise rejected with a "`NotAllowedError`" `DOMException`.
5. If this.`[[sharePromise]]` is not `null`, return a promise rejected with an "`InvalidStateError`" `DOMException`.
6. If *global* does not have transient activation, return a promise rejected with a "`NotAllowedError`" `DOMException`.
7. Consume user activation of *global*.

8. Let *base* be this's relevant settings object's API base URL.
9. If validate share data with *data* and *base* returns false, then return a promise rejected with a `TypeError`.
10. If *data*'s `url` member is present:
    1. Let *url* be the result of running the URL parser on *data*'s `url` with *base*.
    2. Assert: *url* is `URL`.
    3. Set *data* to a copy of *data*, with its `url` member set to the result of running the URL serializer on *url*.
11. If a file type is being blocked due to security considerations, return a promise rejected with a "`NotAllowedError`" `DOMException`.
12. Set this.`[[sharePromise]]` to be a new promise.
13. Return this.`[[sharePromise]]` and in parallel:
    1. If there are no share targets available, queue a global task on the user interaction task source using *global* to:
        1. Reject this.`[[sharePromise]]` with an "`AbortError`" `DOMException`.
        2. Set this.`[[sharePromise]]` to `null`.
        3. Terminate this algorithm.
    2. Present the user with a choice of one more share targets and the ability abort the operation. This UI surface serves as a security confirmation, ensuring that websites cannot silently send data to native applications. The user agent *SHOULD* show intermediary UI through which the user can verify the shared content (if the OS-level UI does not provide this functionality).
    3. Wait for the user's choice.
    4. If the user chose to abort the share operation, queue a global task on the user interaction task source using *global* to:
        1. Reject this.`[[sharePromise]]` with an "`AbortError`" `DOMException`,
        2. Set this.`[[sharePromise]]` to `null`.
        3. Terminate this algorithm.
    5. Activate the chosen share target, convert *data* to a format suitable for ingestion into the target, and transmit the converted data to the target.
    6. If an error occurs starting the target or transmitting the data, queue a global task on the user interaction task source using *global* to:

1. Reject this.`[[sharePromise]]` with an "`DataError`" `DOMException`.
2. Set this.`[[sharePromise]]` to `null`.
3. Terminate this algorithm.

7. Once the data has either been successfully transmitted to the share target, or successfully transmitted to the OS (if the transmission to the share target cannot be confirmed), queue a global task on the user interaction task source using *global* to:

NOTE
Once a share target has been given the payload, the share is considered successful. If the target considers the data unacceptable or an error occurs, it can either recover gracefully, or show an error message to the end-user; it cannot rely on the sender to handle errors. In other words, the `share()` method is "fire and forget"; it does not wait for the target to approve or reject the payload.

1. Resolve this.`[[sharePromise]]` with `undefined`.
2. Set this.`[[sharePromise]]` to `null`.

§ 2.1.3 `canShare(data)` method

▶ MDN ✅

When the ***`canShare()`*** method is called with argument `ShareData` *data*, run the following steps:

1. Let *document* be the this's relevant global object's associated `Document`.
2. If *document* is not fully active, return false.
3. If *document* is not allowed to use `"web-share"`, return false.
4. Return the result of validate share data with *data* and this's relevant settings object's API base URL.

NOTE: canShare() is not future compatible

The `canShare()` method of the Web Share API may return false positive results when used with objects that contain members not defined in the `ShareData` dictionary. Because of how WebIDL works, browsers drop unknown members from the input object and will return `true` when testing the remaining members. This behavior is not future-compatible.

To ensure reliable testing of shareability, developers can construct the input object using only members defined in the `ShareData` dictionary. If an object contains additional members, it should be destructured to only test the defined members individually.

Although this doesn't affect any implementations today, the Working Group will work towards addressing this issue in a future revision of the specification - especially if we make any amendments to the `ShareData` dictionary. To contribute ideas for how to address this issue, see issue #108 on GitHub.

§ 2.1.4 Validate share data

A ***sharable scheme*** is any of the following URL schemes:

- `http`
- `https`
- Any safelisted scheme that the user agent supports for the purpose of sharing.

To ***validate share data*** with *data* and *base*, run the following steps:

1. If none of *data*'s members `title`, `text`, or `url` or `files` are present, return false.
2. Let *titleTextOrUrl* be true if any of `title`, or `text`, or `url` is present.
3. If *data*'s `files` member is present:
    1. If *titleTextOrUrl* is false and *data*'s `files` member is empty, return false.

    NOTE

    This causes a `{ files: [] }` dictionary to be treated as an empty dictionary. However, passing a dictionary like `{text: "text", files: []}` is fine, as `files` is just ignored.

    2. If the implementation does not support file sharing, return false.
    3. If the user agent believes sharing any of the files in `files` would result in a potentially hostile share (i.e., the user agent determines a file is malicious in some way, because of its contents, size, or other characteristic), return false.
4. If *data*'s `url` member is present:
    1. Let *url* be the result of running the URL parser on *data*'s `url` member, with *base*, and no encoding override.
    2. If *url* is failure, return false.
    3. If the *url*'s scheme is a local scheme, or `file`, or `javascript`, or `ws`, or `wss`, return false.
    4. If *url*'s scheme is not a sharable scheme, return false.
5. Return true.

§ 2.2 `ShareData` dictionary

WebIDL

```
dictionary ShareData {
  sequence<File> files;
  USVString title;
  USVString text;
  USVString url;
};
```

The ***`ShareData`*** dictionary consists of several optional members:

`files` **member**
: Files to be shared.

`text` **member**
: Arbitrary text that forms the body of the message being shared.

`title` **member**
: The title of the document being shared. May be ignored by the target.

`url` **member**
: A URL string referring to a resource being shared.

NOTE
These members are `USVString` (as opposed to `DOMString`) because they are not allowed to contain surrogate code points. Among other things, this means that the user agent can serialize them into any Unicode encoding, such as UTF-8, without change or loss of data or the generation of replacement characters.

§ 3. Share targets

A ***share target*** is the abstract concept of a destination that the user agent will transmit the share data to. What constitutes a share target is at the discretion of the user agent.

A share target might not be directly able to accept a `ShareData` (due to not having been written with this API in mind). However, it *MUST* have the ability to receive data that matches some or all of the concepts exposed in `ShareData`. To ***convert data to a format suitable for ingestion into the target***, the user agent *SHOULD* map the members of `ShareData` onto equivalent concepts in the target. It *MAY* discard or combine members if necessary. The meaning of each member of the payload is at the discretion of the share target.

NOTE
Mapping the `ShareData` to the share target's (or operating system's) native format can be tricky as some platforms will not have an equivalent set of members. For example, if the target has a "text" member but not a "URL" member (as is the case on Android), one solution is to concatenate both the `text` and `url` members of `ShareData` and pass the result in the "text" member of the target.

Each share target *MAY* be made conditionally available depending on the `ShareData` payload delivered to the `share()` method.

§ 3.1 Examples of share targets

This section is non-normative.

The list of share targets can be populated from a variety of sources, depending on the user agent and host operating system. For example:

- Built-in service (e.g., "copy to clipboard").

- Native applications written for the host operating system.
- Contacts (e.g., the user agent directly shares to a person from the user's address book, using a specific app).
- Websites (e.g., the user agent fills in a template URL with the members of the `ShareData`, then navigates to that URL in a new browsing context).

> NOTE
> There is an attempt to standardize the registration of websites to receive share data for that final use case; see Web Share Target.

In some cases, the host operating system will provide a sharing or intent system similar to Web Share. In these cases, the user agent can simply forward the share data to the operating system and not talk directly to native applications.

§ 4. Permissions Policy integration

▸ MDN 🚫

▸ MDN 🚫

This specification defines a policy-controlled permission identified by the string **"*web-share*"**. Its default allowlist is 'self', which means third-party contexts are not allowed to use the API by default.

It is *OPTIONAL* for user agents to support *Permissions Policy*'s `Permissions-Policy` HTTP header.

Developers can use the means afforded by the *Permissions Policy* specification to control if and when a third-party context is allowed to use this API.

> NOTE: Permissions Policy Implementation Status
>
> Although user agents are unified in preventing the Web Share API from being used in third-party context, at the time of publication there are interoperability issues with relying on the *Permissions Policy* to enable the API in third-party contexts. In particular, although the`allow` attribute is widely supported, the updated syntax for the `allow` attribute is not. Similarly, the `Permissions-Policy:` HTTP header is not yet widely supported. Developers are advised to check the implementation status of the *Permissions Policy* specification before relying on it to enable the Web Share API in third-party contexts.
>
> When the `Permissions-Policy` HTTP header is more widely supported, the Working Group expects to revise this specification to require support for the header.

§ 5. Accessibility considerations

This section is non-normative.

When this specification is used to present information in the user interface, implementors will want to follow the OS level accessibility guidelines for the platform. Further, as sharing can have potential security implications for end-users, as outlined as part of the `share()` method, the share UI needs to be presented in an accessible manner, while also taking into consideration a platform's security guidelines for user interfaces. Some key considerations are to:

- Provide alternative text for images and other non-textual content to ensure that users who are visually impaired can understand the type and/or nature of the content being shared.
- Ensure that users can navigate through the previewed content using keyboard-only controls, and that the keyboard focus is clearly visible.
- Use headings and labels to ensure that the previewed content is properly structured and can be understood by users using assistive technology.
- Ensure sufficient color contrast and font size, and use fonts, icons, and images that are widely recognised and easily understood, in the share UI.

Together, these elements can help users with a range of visual, motor or cognitive disabilities to better understand the nature of the content being shared by a web page.

§ 6. Privacy considerations

- By design, the API cannot be used by a website to learn which share targets are available, or which share target the user chose from `share()`. This is to prevent leaking information that could be used for fingerprinting, as well as leaking details about the user's device or user's preferred share targets.
- Use of `share()` from a private browsing mode might leak private data to a third-party application that does not respect the user's privacy setting. User agents could present additional warnings or *MAY* disable the feature entirely when in a private browsing mode, but this is not mandated as the chooser UI could be considered sufficient warning.

§ 7. Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words *MAY*, *MUST*, *OPTIONAL*, and *SHOULD* in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

§ 8. IDL Index

WebIDL

```
partial interface Navigator {
  [SecureContext] Promise<undefined> share(optional ShareData data =
{});
  [SecureContext] boolean canShare(optional ShareData data = {});
};

dictionary ShareData {
  sequence<File> files;
  USVString title;
  USVString text;
  USVString url;
};
```

§ 9. Changelog

This section is non-normative.

The following normative changes were made to the specification during the Proposed Recommendation phase. Please see the commit log for a complete list of changes.

- Make Permissions-Policy HTTP header OPTIONAL (#275)

The following normative changes were made to the specification since it was published as a First Public Working Draft to Candidate Recommendation. Please see the commit log for a complete list of changes.

- Move Priv/Sec into spec (#245)
- Define sharable scheme + check (#244)
- Handle non-fully-active documents (#219)
- Add canShare() method (#177)
- Add "validate share data" algorithm (#185)
- Resolve promise after OS hand-off (#209)
- Check transient activation after sharePromise (#183)
- Reject non-http URLs for url member (#174)
- Use a sequence for files (#170)
- Introduce "web-share" feature policy (#166)
- BREAKING CHANGE: make share() consume user activation (#137)
- Add the ability to share files (#133)
- Accessibility considerations (#146)

§ A. Acknowledgments

The editors would like to thank the following W3C groups for their invaluable feedback, which greatly improved this specification: Accessible Platform Architectures Working Group, the Internationalization Working Group, the Privacy Interest Group, and the Technical Architecture Group.

Thanks to the Web Intents team, who laid the groundwork for the web app interoperability use cases. In particular, Paul Kinlan, who did a lot of early advocacy for Web Share.

§ B. References

§ B.1 Normative references

[fetch]
Fetch Standard. Anne van Kesteren. WHATWG. Living Standard. URL: https://fetch.spec.whatwg.org/

[fileapi]
File API. Marijn Kruisselbrink. W3C. 6 February 2023. W3C Working Draft. URL: https://www.w3.org/TR/FileAPI/

[html]
HTML Standard. Anne van Kesteren; Domenic Denicola; Ian Hickson; Philip Jägenstedt; Simon Pieters. WHATWG. Living Standard. URL: https://html.spec.whatwg.org/multipage/

[PERMISSIONS-POLICY]
Permissions Policy. Ian Clelland. W3C. 22 March 2023. W3C Working Draft. URL: https://www.w3.org/TR/permissions-policy-1/

[RFC2119]
Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. IETF. March 1997. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc2119

[RFC8174]
Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words. B. Leiba. IETF. May 2017. Best Current Practice. URL: https://www.rfc-editor.org/rfc/rfc8174

[url]
URL Standard. Anne van Kesteren. WHATWG. Living Standard. URL: https://url.spec.whatwg.org/

[WEBIDL]
Web IDL Standard. Edgar Chen; Timothy Gu. WHATWG. Living Standard. URL: https://webidl.spec.whatwg.org/

§ B.2 Informative references

[encoding]
Encoding Standard. Anne van Kesteren. WHATWG. Living Standard. URL: https://encoding.spec.whatwg.org/