dialog

> Display native system dialogs for opening and saving files, alerting, etc.

Process: **Main**

An example of showing a dialog to select multiple files:

```
const { dialog } = require('electron')
console.log(dialog.showOpenDialog({ properties: ['openFile', 'multiSelections']
}))
```

Methods

The `dialog` module has the following methods:

`dialog.showOpenDialogSync([window, ]options)`

- `window` **BaseWindow** (optional)
- `options` Object
  - `title` string (optional)
  - `defaultPath` string (optional)
  - `buttonLabel` string (optional) - Custom label for the confirmation button, when left empty the default label will be used.
  - `filters` **FileFilter[]** (optional)
  - `properties` string[] (optional) - Contains which features the dialog should use. The following values are supported:
    - `openFile` - Allow files to be selected.
    - `openDirectory` - Allow directories to be selected.
    - `multiSelections` - Allow multiple paths to be selected.
    - `showHiddenFiles` - Show hidden files in dialog.

- `createDirectory` *macOS* - Allow creating new directories from dialog.
- `promptToCreate` *Windows* - Prompt for creation if the file path entered in the dialog does not exist. This does not actually create the file at the path but allows non-existent paths to be returned that should be created by the application.
- `noResolveAliases` *macOS* - Disable the automatic alias (symlink) path resolution. Selected aliases will now return the alias path instead of their target path.
- `treatPackageAsDirectory` *macOS* - Treat packages, such as `.app` folders, as a directory instead of a file.
- `dontAddToRecent` *Windows* - Do not add the item being opened to the recent documents list.

- `message` string (optional) *macOS* - Message to display above input boxes.
- `securityScopedBookmarks` boolean (optional) *macOS* *MAS* - Create **security scoped bookmarks** when packaged for the Mac App Store.

Returns `string[] | undefined`, the file paths chosen by the user; if the dialog is cancelled it returns `undefined`.

The `window` argument allows the dialog to attach itself to a parent window, making it modal.

The `filters` specifies an array of file types that can be displayed or selected when you want to limit the user to a specific type. For example:

```
{
  filters: [
    { name: 'Images', extensions: ['jpg', 'png', 'gif'] },
    { name: 'Movies', extensions: ['mkv', 'avi', 'mp4'] },
    { name: 'Custom File Type', extensions: ['as'] },
    { name: 'All Files', extensions: ['*'] }
  ]
}
```

The `extensions` array should contain extensions without wildcards or dots (e.g. `'png'` is good but `'.png'` and `'*.png'` are bad). To show all files, use the `'*'` wildcard (no other wildcard is supported).

> **NOTE**

> On Windows and Linux an open dialog can not be both a file selector and a directory selector, so if you set `properties` to `['openFile', 'openDirectory']` on these platforms, a directory selector will be shown.

```
dialog.showOpenDialogSync(mainWindow, {
  properties: ['openFile', 'openDirectory']
})
```

> **NOTE**
>
> On Linux `defaultPath` is not supported when using portal file chooser dialogs unless the portal backend is version 4 or higher. You can use `--xdg-portal-required-version` **command-line switch** to force gtk or kde dialogs.

`dialog.showOpenDialog([window, ]options)`

- `window` **BaseWindow** (optional)
- `options` Object
  - `title` string (optional)
  - `defaultPath` string (optional)
  - `buttonLabel` string (optional) - Custom label for the confirmation button, when left empty the default label will be used.
  - `filters` **FileFilter[]** (optional)
  - `properties` string[] (optional) - Contains which features the dialog should use. The following values are supported:
    - `openFile` - Allow files to be selected.
    - `openDirectory` - Allow directories to be selected.
    - `multiSelections` - Allow multiple paths to be selected.
    - `showHiddenFiles` - Show hidden files in dialog.
    - `createDirectory` *macOS* - Allow creating new directories from dialog.
    - `promptToCreate` *Windows* - Prompt for creation if the file path entered in the dialog does not exist. This does not actually create the file at the path but allows non-existent paths to be returned that should be created by the application.

        - `noResolveAliases` *macOS* - Disable the automatic alias (symlink) path resolution. Selected aliases will now return the alias path instead of their target path.
        - `treatPackageAsDirectory` *macOS* - Treat packages, such as `.app` folders, as a directory instead of a file.
        - `dontAddToRecent` *Windows* - Do not add the item being opened to the recent documents list.
    - `message` string (optional) *macOS* - Message to display above input boxes.
    - `securityScopedBookmarks` boolean (optional) *macOS* *MAS* - Create **security scoped bookmarks** when packaged for the Mac App Store.

Returns `Promise<Object>` - Resolve with an object containing the following:

- `canceled` boolean - whether or not the dialog was canceled.
- `filePaths` string[] - An array of file paths chosen by the user. If the dialog is cancelled this will be an empty array.
- `bookmarks` string[] (optional) *macOS* *MAS* - An array matching the `filePaths` array of base64 encoded strings which contains security scoped bookmark data. `securityScopedBookmarks` must be enabled for this to be populated. (For return values, see **table here**.)

The `window` argument allows the dialog to attach itself to a parent window, making it modal.

The `filters` specifies an array of file types that can be displayed or selected when you want to limit the user to a specific type. For example:

```
{
  filters: [
    { name: 'Images', extensions: ['jpg', 'png', 'gif'] },
    { name: 'Movies', extensions: ['mkv', 'avi', 'mp4'] },
    { name: 'Custom File Type', extensions: ['as'] },
    { name: 'All Files', extensions: ['*'] }
  ]
}
```

The `extensions` array should contain extensions without wildcards or dots (e.g. `'png'` is good but `'.png'` and `'*.png'` are bad). To show all files, use the `'*'` wildcard (no other wildcard is

supported).

> **NOTE**
>
> On Windows and Linux an open dialog can not be both a file selector and a directory selector, so if you set `properties` to `['openFile', 'openDirectory']` on these platforms, a directory selector will be shown.

```
dialog.showOpenDialog(mainWindow, {
  properties: ['openFile', 'openDirectory']
}).then(result => {
  console.log(result.canceled)
  console.log(result.filePaths)
}).catch(err => {
  console.log(err)
})
```

> **NOTE**
>
> On Linux `defaultPath` is not supported when using portal file chooser dialogs unless the portal backend is version 4 or higher. You can use `--xdg-portal-required-version` **command-line switch** to force gtk or kde dialogs.

`dialog.showSaveDialogSync([window, ]options)`

- `window` **BaseWindow** (optional)
- `options` Object
  - `title` string (optional) - The dialog title. Cannot be displayed on some *Linux* desktop environments.
  - `defaultPath` string (optional) - Absolute directory path, absolute file path, or file name to use by default.
  - `buttonLabel` string (optional) - Custom label for the confirmation button, when left empty the default label will be used.
  - `filters` **FileFilter[]** (optional)
  - `message` string (optional) *macOS* - Message to display above text fields.

- `nameFieldLabel` string (optional) *macOS* - Custom label for the text displayed in front of the filename text field.
- `showsTagField` boolean (optional) *macOS* - Show the tags input box, defaults to `true`.
- `properties` string[] (optional)
  - `showHiddenFiles` - Show hidden files in dialog.
  - `createDirectory` *macOS* - Allow creating new directories from dialog.
  - `treatPackageAsDirectory` *macOS* - Treat packages, such as `.app` folders, as a directory instead of a file.
  - `showOverwriteConfirmation` *Linux* - Sets whether the user will be presented a confirmation dialog if the user types a file name that already exists.
  - `dontAddToRecent` *Windows* - Do not add the item being saved to the recent documents list.
- `securityScopedBookmarks` boolean (optional) *macOS* *MAS* - Create a **security scoped bookmark** when packaged for the Mac App Store. If this option is enabled and the file doesn't already exist a blank file will be created at the chosen path.

Returns `string`, the path of the file chosen by the user; if the dialog is cancelled it returns an empty string.

The `window` argument allows the dialog to attach itself to a parent window, making it modal.

The `filters` specifies an array of file types that can be displayed, see `dialog.showOpenDialog` for an example.

`dialog.showSaveDialog([window, ]options)`

- `window` **BaseWindow** (optional)
- `options` Object
  - `title` string (optional) - The dialog title. Cannot be displayed on some *Linux* desktop environments.
  - `defaultPath` string (optional) - Absolute directory path, absolute file path, or file name to use by default.
  - `buttonLabel` string (optional) - Custom label for the confirmation button, when left empty the default label will be used.
  - `filters` **FileFilter[]** (optional)

- `message` string (optional) *macOS* - Message to display above text fields.
- `nameFieldLabel` string (optional) *macOS* - Custom label for the text displayed in front of the filename text field.
- `showsTagField` boolean (optional) *macOS* - Show the tags input box, defaults to `true`.
- `properties` string[] (optional)
  - `showHiddenFiles` - Show hidden files in dialog.
  - `createDirectory` *macOS* - Allow creating new directories from dialog.
  - `treatPackageAsDirectory` *macOS* - Treat packages, such as `.app` folders, as a directory instead of a file.
  - `showOverwriteConfirmation` *Linux* - Sets whether the user will be presented a confirmation dialog if the user types a file name that already exists.
  - `dontAddToRecent` *Windows* - Do not add the item being saved to the recent documents list.
- `securityScopedBookmarks` boolean (optional) *macOS* *MAS* - Create a **security scoped bookmark** when packaged for the Mac App Store. If this option is enabled and the file doesn't already exist a blank file will be created at the chosen path.

Returns `Promise<Object>` - Resolve with an object containing the following:

- `canceled` boolean - whether or not the dialog was canceled.
- `filePath` string - If the dialog is canceled, this will be an empty string.
- `bookmark` string (optional) *macOS* *MAS* - Base64 encoded string which contains the security scoped bookmark data for the saved file. `securityScopedBookmarks` must be enabled for this to be present. (For return values, see **table here**.)

The `window` argument allows the dialog to attach itself to a parent window, making it modal.

The `filters` specifies an array of file types that can be displayed, see `dialog.showOpenDialog` for an example.

> **NOTE**
>
> On macOS, using the asynchronous version is recommended to avoid issues when expanding and collapsing the dialog.

`dialog.showMessageBoxSync([window, ]options)`

- `window` **BaseWindow** (optional)
- `options` Object
  - `message` string - Content of the message box.
  - `type` string (optional) - Can be `none`, `info`, `error`, `question` or `warning`. On Windows, `question` displays the same icon as `info`, unless you set an icon using the `icon` option. On macOS, both `warning` and `error` display the same warning icon.
  - `buttons` string[] (optional) - Array of texts for buttons. On Windows, an empty array will result in one button labeled "OK".
  - `defaultId` Integer (optional) - Index of the button in the buttons array which will be selected by default when the message box opens.
  - `title` string (optional) - Title of the message box, some platforms will not show it.
  - `detail` string (optional) - Extra information of the message.
  - `icon` (**NativeImage** | string) (optional)
  - `textWidth` Integer (optional) *macOS* - Custom width of the text in the message box.
  - `cancelId` Integer (optional) - The index of the button to be used to cancel the dialog, via the `Esc` key. By default this is assigned to the first button with "cancel" or "no" as the label. If no such labeled buttons exist and this option is not set, `0` will be used as the return value.
  - `noLink` boolean (optional) - On Windows Electron will try to figure out which one of the `buttons` are common buttons (like "Cancel" or "Yes"), and show the others as command links in the dialog. This can make the dialog appear in the style of modern Windows apps. If you don't like this behavior, you can set `noLink` to `true`.
  - `normalizeAccessKeys` boolean (optional) - Normalize the keyboard access keys across platforms. Default is `false`. Enabling this assumes `&` is used in the button labels for the placement of the keyboard shortcut access key and labels will be converted so they work correctly on each platform, `&` characters are removed on macOS, converted to `_` on Linux, and left untouched on Windows. For example, a button label of `Vie&w` will be converted to `Vie_w` on Linux and `View` on macOS and can be selected via `Alt-W` on Windows and Linux.

Returns `Integer` - the index of the clicked button.

Shows a message box, it will block the process until the message box is closed. It returns the index of the clicked button.

The `window` argument allows the dialog to attach itself to a parent window, making it modal. If `window` is not shown dialog will not be attached to it. In such case it will be displayed as an independent window.

`dialog.showMessageBox([window, ]options)`

- `window` **BaseWindow** (optional)
- `options` Object
  - `message` string - Content of the message box.
  - `type` string (optional) - Can be `none`, `info`, `error`, `question` or `warning`. On Windows, `question` displays the same icon as `info`, unless you set an icon using the `icon` option. On macOS, both `warning` and `error` display the same warning icon.
  - `buttons` string[] (optional) - Array of texts for buttons. On Windows, an empty array will result in one button labeled "OK".
  - `defaultId` Integer (optional) - Index of the button in the buttons array which will be selected by default when the message box opens.
  - `signal` AbortSignal (optional) - Pass an instance of **AbortSignal** to optionally close the message box, the message box will behave as if it was cancelled by the user. On macOS, `signal` does not work with message boxes that do not have a parent window, since those message boxes run synchronously due to platform limitations.
  - `title` string (optional) - Title of the message box, some platforms will not show it.
  - `detail` string (optional) - Extra information of the message.
  - `checkboxLabel` string (optional) - If provided, the message box will include a checkbox with the given label.
  - `checkboxChecked` boolean (optional) - Initial checked state of the checkbox. `false` by default.
  - `icon` (**NativeImage** | string) (optional)
  - `textWidth` Integer (optional) *macOS* - Custom width of the text in the message box.
  - `cancelId` Integer (optional) - The index of the button to be used to cancel the dialog, via the `Esc` key. By default this is assigned to the first button with "cancel" or "no" as the label. If no such labeled buttons exist and this option is not set, `0` will be used as the return value.
  - `noLink` boolean (optional) - On Windows Electron will try to figure out which one of the `buttons` are common buttons (like "Cancel" or "Yes"), and show the others as command

links in the dialog. This can make the dialog appear in the style of modern Windows apps. If you don't like this behavior, you can set `noLink` to `true`.

- `normalizeAccessKeys` boolean (optional) - Normalize the keyboard access keys across platforms. Default is `false`. Enabling this assumes `&` is used in the button labels for the placement of the keyboard shortcut access key and labels will be converted so they work correctly on each platform, `&` characters are removed on macOS, converted to `_` on Linux, and left untouched on Windows. For example, a button label of `Vie&w` will be converted to `Vie_w` on Linux and `View` on macOS and can be selected via `Alt-W` on Windows and Linux.

Returns `Promise<Object>` - resolves with a promise containing the following properties:

- `response` number - The index of the clicked button.
- `checkboxChecked` boolean - The checked state of the checkbox if `checkboxLabel` was set. Otherwise `false`.

Shows a message box.

The `window` argument allows the dialog to attach itself to a parent window, making it modal.

`dialog.showErrorBox(title, content)`

- `title` string - The title to display in the error box.
- `content` string - The text content to display in the error box.

Displays a modal dialog that shows an error message.

This API can be called safely before the `ready` event the `app` module emits, it is usually used to report errors in early stage of startup. If called before the app `ready` event on Linux, the message will be emitted to stderr, and no GUI dialog will appear.

`dialog.showCertificateTrustDialog([window, ]options)` *macOS* *Windows*

- `window` **BaseWindow** (optional)
- `options` Object
  - `certificate` **Certificate** - The certificate to trust/import.
  - `message` string - The message to display to the user.

Returns `Promise<void>` - resolves when the certificate trust dialog is shown.

On macOS, this displays a modal dialog that shows a message and certificate information, and gives the user the option of trusting/importing the certificate. If you provide a `window` argument the dialog will be attached to the parent window, making it modal.

On Windows the options are more limited, due to the Win32 APIs used:

- The `message` argument is not used, as the OS provides its own confirmation dialog.
- The `window` argument is ignored since it is not possible to make this confirmation dialog modal.

Bookmarks array

`showOpenDialog` and `showSaveDialog` resolve to an object with a `bookmarks` field. This field is an array of Base64 encoded strings that contain the **security scoped bookmark** data for the saved file. The `securityScopedBookmarks` option must be enabled for this to be present.

Build Type	securityScopedBookmarks boolean	Return Type	Return Value
macOS mas	True	Success	`['LONGBOOKMARKSTRING']`
macOS mas	True	Error	`['']` (array of empty string)
macOS mas	False	NA	`[]` (empty array)
non mas	any	NA	`[]` (empty array)

Sheets

On macOS, dialogs are presented as sheets attached to a window if you provide a **`BaseWindow`** reference in the `window` parameter, or modals if no window is provided.

You can call `BaseWindow.getCurrentWindow().setSheetOffset(offset)` to change the offset from the window frame where sheets are attached.

Edit this page