Thunderbird WebExtensions Documentation

Release latest

Jan 15, 2020

Contents

Thunderbird WebExtensions are very similar to those of Firefox. These documents assume you have some familiarity with building a WebExtension for Firefox. If not, it is highly recommended to begin by reading some of the MDN documentation on the subject.

WebExtension APIs are asynchronous, that is, they return a Promise object which resolves when ready. See Using Promises for more information about Promises.

The documents were generated automatically from the schema documents at mail/components/extensions/schemas.

Note: This documentation is for pre-release versions of Thunderbird. See the "68" version for Thunderbird 68.

Note: These APIs should be considered experimental and may change in the future. For any problems or feature requests please file a bug.

CHAPTER 1

accounts

The accounts API first appeared in Thunderbird 66 (see bug 1488176).

1.1 Permissions

- accountsRead "See your mail accounts and their folders"

Note: The permission `accountsRead` is required to use `accounts`.

1.2 Functions

1.2.1 list()

Returns all mail accounts.

Returns a Promise fulfilled with:

- array of *MailAccount*

1.2.2 get(accountId)

Returns details of the requested account, or null if it doesn't exist.

- `accountId` (string)

Returns a Promise fulfilled with:

- *MailAccount*

1.3 Types

1.3.1 MailAccount

object

- `folders` (array of *MailFolder*) The folders for this account.
- `id` (string) A unique identifier for this account.
- `name` (string) The human-friendly name of this account.
- `type` (string) What sort of account this is, e.g. `imap`, `nntp`, or `pop3`.

CHAPTER 2

addressBooks

The address books API, also including the *contacts* and *mailingLists* namespaces, first appeared in Thunderbird 64.

The Address Books sample extension uses this API.

2.1 Permissions

- addressBooks "Read and modify your address books and contacts"

Note: The permission `addressBooks` is required to use `addressBooks`.

2.2 Functions

2.2.1 openUI()

Opens the address book user interface.

2.2.2 closeUI()

Closes the address book user interface.

2.2.3 list([complete])

Gets a list of the user's address books, optionally including all contacts and mailing lists.

- `[complete]` (boolean) If set to true, results will include contacts and mailing lists for each address book.

Returns a Promise fulfilled with:

- array of *AddressBookNode*

2.2.4 get(id, [complete])

Gets a single address book, optionally including all contacts and mailing lists.

- `id` (string)
- [`complete`] (boolean) If set to true, results will include contacts and mailing lists for this address book.

Returns a Promise fulfilled with:

- *AddressBookNode*

2.2.5 create(properties)

Creates a new, empty address book.

- `properties` (object)
  - `name` (string)

Returns a Promise fulfilled with:

- string The ID of the new address book.

2.2.6 update(id, properties)

Renames an address book.

- `id` (string)
- `properties` (object)
  - `name` (string)

2.2.7 delete(id)

Removes an address book, and all associated contacts and mailing lists.

- `id` (string)

2.3 Events

2.3.1 onCreated(node)

Fired when an address book is created.

- `node` (*AddressBookNode*)

2.3.2 onUpdated(node)

Fired when an address book is renamed.

- `node` (*AddressBookNode*)

2.3.3 onDeleted(id)

Fired when an addressBook is deleted.

- `id` (string)

2.4 Types

2.4.1 AddressBookNode

A node representing an address book.

object

- `id` (string) The unique identifier for the node. IDs are unique within the current profile, and they remain valid even after the program is restarted.
- `name` (string)
- `type` (*NodeType*) Always set to `addressBook`.
- [`contacts`] (array of *ContactNode*) A list of contacts held by this node's address book or mailing list.
- [`mailingLists`] (array of *MailingListNode*) A list of mailingLists in this node's address book.
- [`parentId`] (string) The `id` of the parent object.
- [`readOnly`] (boolean) Indicates if the object is read-only. Currently this returns false in all cases, as read-only address books are ignored by the API.

2.4.2 NodeType

Indicates the type of a Node, which can be one of `addressBook`, `contact`, or `mailingList`.

string

Values for NodeType:

- `addressBook`
- `contact`
- `mailingList`

CHAPTER 3

browserAction

The browserAction and *composeAction* APIs first appeared in Thunderbird 64. They are very similar to Firefox's browserAction API.

Many of our sample extensions use a browserAction.

Use toolbar actions to put icons in the mail window toolbar. In addition to its icon, a toolbar action can also have a tooltip, a badge, and a popup. This namespace is called browserAction for compatibility with browser WebExtensions.

3.1 Manifest file properties

- [`browser_action`] (object)
  - [`browser_style`] (boolean)
  - [`default_area`] (string) Currently unused.
  - [`default_icon`] (IconPath)
  - [`default_popup`] (string)
  - [`default_title`] (string)
  - [`theme_icons`] (array of ThemeIcons) Specifies icons to use for dark and light themes

Note: A manifest entry named `browser_action` is required to use `browserAction`.

3.2 Functions

3.2.1 setTitle(details)

Sets the title of the toolbar action. This shows up in the tooltip.

- `details` (object)
  - `title` (string or null) The string the toolbar action should display when moused over.

3.2.2 getTitle(details)

Gets the title of the toolbar action.

- `details` (*Details*)

Returns a Promise fulfilled with:

- string

3.2.3 setIcon(details)

Sets the icon for the toolbar action. The icon can be specified either as the path to an image file or as the pixel data from a canvas element, or as dictionary of either one of those. Either the **path** or the **imageData** property must be specified.

- `details` (object)
  - [`imageData`] (*ImageDataType* or object) Either an ImageData object or a dictionary {size -> ImageData} representing icon to be set. If the icon is specified as a dictionary, the actual image to be used is chosen depending on screen's pixel density. If the number of image pixels that fit into one screen space unit equals `scale`, then image with size `scale` * 19 will be selected. Initially only scales 1 and 2 will be supported. At least one image must be specified. Note that 'details.imageData = foo' is equivalent to 'details.imageData = {'19': foo}'
  - [`path`] (string or object) Either a relative image path or a dictionary {size -> relative image path} pointing to icon to be set. If the icon is specified as a dictionary, the actual image to be used is chosen depending on screen's pixel density. If the number of image pixels that fit into one screen space unit equals `scale`, then image with size `scale` * 19 will be selected. Initially only scales 1 and 2 will be supported. At least one image must be specified. Note that 'details.path = foo' is equivalent to 'details.imageData = {'19': foo}'

3.2.4 setPopup(details)

Sets the html document to be opened as a popup when the user clicks on the toolbar action's icon.

- `details` (object)
  - `popup` (string or null) The html file to show in a popup. If set to the empty string (''), no popup is shown.

3.2.5 getPopup(details)

Gets the html document set as the popup for this toolbar action.

- `details` (*Details*)

Returns a Promise fulfilled with:

- string

3.2.6 setBadgeText(details)

Sets the badge text for the toolbar action. The badge is displayed on top of the icon.

- `details` (object)
  - `text` (string or null) Any number of characters can be passed, but only about four can fit in the space.

3.2.7 getBadgeText(details)

Gets the badge text of the toolbar action. If no tab nor window is specified is specified, the global badge text is returned.

- `details` (*Details*)

Returns a Promise fulfilled with:

- string

3.2.8 setBadgeBackgroundColor(details)

Sets the background color for the badge.

- `details` (object)
  - `color` (string or *ColorArray* or null) An array of four integers in the range [0,255] that make up the RGBA color of the badge. For example, opaque red is `[255, 0, 0, 255]`. Can also be a string with a CSS value, with opaque red being `#FF0000` or `#F00`.

3.2.9 getBadgeBackgroundColor(details)

Gets the background color of the toolbar action.

- `details` (*Details*)

Returns a Promise fulfilled with:

- *ColorArray*

3.2.10 enable([tabId])

Enables the toolbar action for a tab. By default, toolbar actions are enabled.

- [`tabId`] (integer) The id of the tab for which you want to modify the toolbar action.

3.2.11 disable([tabId])

Disables the toolbar action for a tab.

- [`tabId`] (integer) The id of the tab for which you want to modify the toolbar action.

3.2.12 isEnabled(details)

Checks whether the toolbar action is enabled.

- `details` (*Details*)

3.2.13 openPopup()

Opens the extension popup window in the active window.

3.3 Events

3.3.1 onClicked()

Fired when a toolbar action icon is clicked. This event will not fire if the toolbar action has a popup.

3.4 Types

3.4.1 ColorArray

array of integer

3.4.2 Details

Specifies to which tab or window the value should be set, or from which one it should be retrieved. If no tab nor window is specified, the global value is set or retrieved.

object

- [`tabId`] (integer) When setting a value, it will be specific to the specified tab, and will automatically reset when the tab navigates. When getting, specifies the tab to get the value from; if there is no tab-specific value, the window one will be inherited.
- [`windowId`] (integer) When setting a value, it will be specific to the specified window. When getting, specifies the window to get the value from; if there is no window-specific value, the global one will be inherited.

3.4.3 ImageDataType

Pixel data for an image. Must be an ImageData object (for example, from a `canvas` element).

ImageData

CHAPTER 4

cloudFile

The cloudFile (a.k.a. fileLink) API first appeared in Thunderbird 64, and was uplifted to Thunderbird 60.4 ESR.

From Thunderbird 68.2.1 (Thunderbird 71 beta), an extension can choose to receive data for upload as a `File` object rather than as an `ArrayBuffer`. You **should** specify which you want as the default may change in a future version.

The DropBox Uploader sample extension uses this API.

4.1 Manifest file properties

- [`cloud_file`] (object)
  - `management_url` (string) A page for configuring accounts, to be displayed in the preferences UI.
  - `name` (string) Name of the cloud file service.
  - [`data_format`] (*string*) Determines the format of the `data` argument in `onFileUpload`. *Added in Thunderbird 71, backported to 68.2.1*
  - [`new_account_url`] (string) **Deprecated.** This property was never used.
  - [`service_url`] (string) URL to the web page of the cloud file service.

Values for data_format:

- `ArrayBuffer`
- `File`

Note: A manifest entry named `cloud_file` is required to use `cloudFile`.

4.2 Functions

4.2.1 getAccount(accountId)

Retrieve information about a single cloud file account

- `accountId` (string) Unique identifier of the account

Returns a Promise fulfilled with:

- *CloudFileAccount*

4.2.2 getAllAccounts()

Retrieve all cloud file accounts for the current add-on

Returns a Promise fulfilled with:

- array of *CloudFileAccount*

4.2.3 updateAccount(accountId, updateProperties)

Update a cloud file account

- `accountId` (string) Unique identifier of the account
- `updateProperties` (object)
  - [`configured`] (boolean) If true, the account is configured and ready to use. This property is currently ignored and all accounts are assumed to be configured.
  - [`managementUrl`] (string) A page for configuring accounts, to be displayed in the preferences UI.
  - [`spaceRemaining`] (integer) The amount of remaining space on the cloud provider, in bytes. Set to -1 if unsupported.
  - [`spaceUsed`] (integer) The amount of space already used on the cloud provider, in bytes. Set to -1 if unsupported.
  - [`uploadSizeLimit`] (integer) The maximum size in bytes for a single file to upload. Set to -1 if unlimited.

Returns a Promise fulfilled with:

- *CloudFileAccount*

4.3 Events

4.3.1 onFileUpload(account, fileInfo)

Fired when a file should be uploaded to the cloud file provider

- `account` (*CloudFileAccount*) The created account
- `fileInfo` (*CloudFile*) The file to upload

Event listeners should return:

- object
  - [aborted] (boolean) Set this to true if the file upload was aborted
  - [url] (string) The URL where the uploaded file can be accessed

4.3.2 onFileUploadAbort(account, fileId)

- account (*CloudFileAccount*) The created account
- fileId (integer) An identifier for this file

4.3.3 onFileDeleted(account, fileId)

Fired when a file previously uploaded should be deleted

- account (*CloudFileAccount*) The created account
- fileId (integer) An identifier for this file

4.3.4 onAccountAdded(account)

Fired when a cloud file account of this add-on was created

- account (*CloudFileAccount*) The created account

4.3.5 onAccountDeleted(accountId)

Fired when a cloud file account of this add-on was deleted

- accountId (string) The id of the removed account

4.4 Types

4.4.1 CloudFile

Information about a cloud file

object

- data (ArrayBuffer or File)
- id (integer) An identifier for this file
- name (string) Filename of the file to be transferred

4.4.2 CloudFileAccount

Information about a cloud file account

object

- configured (boolean) If true, the account is configured and ready to use. This property is currently ignored and all accounts are assumed to be configured.

- `id` (string) Unique identifier of the account
- `managementUrl` (string) A page for configuring accounts, to be displayed in the preferences UI.
- `name` (string) A user-friendly name for this account.
- [`spaceRemaining`] (integer) The amount of remaining space on the cloud provider, in bytes. Set to -1 if unsupported.
- [`spaceUsed`] (integer) The amount of space already used on the cloud provider, in bytes. Set to -1 if unsupported.
- [`uploadSizeLimit`] (integer) The maximum size in bytes for a single file to upload. Set to -1 if unlimited.

CHAPTER 5

commands

The commands API first appeared in Thunderbird 66. It's more or less the same as the Firefox commands API.

Use the commands API to add keyboard shortcuts that trigger actions in your extension, for example, an action to open the browser action or send a command to the xtension.

5.1 Manifest file properties

- [`commands`] (object)

Note: A manifest entry named `commands` is required to use `commands`.

5.2 Functions

5.2.1 update(detail)

Update the details of an already defined command.

- `detail` (object) The new description for the command.
  - `name` (string) The name of the command.
  - [`description`] (string) The new description for the command.
  - [`shortcut`] (manifest.KeyName)

5.2.2 reset(name)

Reset a command's details to what is specified in the manifest.

- `name` (string) The name of the command.

5.2.3 getAll()

Returns all the registered extension commands for this extension and their shortcut (if active).

Returns a Promise fulfilled with:

- array of *Command*

5.3 Events

5.3.1 onCommand(command)

Fired when a registered command is activated using a keyboard shortcut.

- `command` (string)

5.4 Types

5.4.1 Command

object

- [`description`] (string) The Extension Command description
- [`name`] (string) The name of the Extension Command
- [`shortcut`] (string) The shortcut active for this command, or blank if not active.

CHAPTER 6

compose

This message composition window API first appeared in Thunderbird 67 (see bug 1503423).

6.1 Functions

6.1.1 beginNew([details])

- [details] (*ComposeParams*)

6.1.2 beginReply(messageId, [replyType])

- messageId (integer) The message to reply to, as retrieved using other APIs.
- [replyType] (*string*)

Values for replyType:

- replyToSender
- replyToList
- replyToAll

6.1.3 beginForward(messageId, [forwardType], [details])

- messageId (integer) The message to forward, as retrieved using other APIs.
- [forwardType] (*string*)
- [details] (*ComposeParams*)

Values for forwardType:

- forwardInline

- `forwardAsAttachment`

6.2 Types

6.2.1 ComposeParams

object

- [`bcc`] (array of *ComposeRecipient*)
- [`body`] (string)
- [`cc`] (array of *ComposeRecipient*)
- [`replyTo`] (string)
- [`subject`] (string)
- [`to`] (array of *ComposeRecipient*)

6.2.2 ComposeRecipient

string: A name and email address in the format "Name <email@example.com>", or just an email address.

OR

object:

- `id` (string) The ID of a contact or mailing list from the *contacts* and *mailingLists* APIs.
- `type` (*string*) Which sort of object this ID is for.

Values for type:

- `contact`
- `mailingList`

CHAPTER 7

composeAction

The *browserAction* and composeAction APIs first appeared in Thunderbird 64. They are very similar to Firefox's browserAction API.

Use toolbar actions to put icons in the message composition toolbars. In addition to its icon, a toolbar action can also have a tooltip, a badge, and a popup.

7.1 Manifest file properties

- [`compose_action`] (object)
  - [`browser_style`] (boolean)
  - [`default_area`] (*string*) Defines the location the composeAction will appear by default. The default location is maintoolbar.
  - [`default_icon`] (IconPath)
  - [`default_popup`] (string)
  - [`default_title`] (string)
  - [`theme_icons`] (array of ThemeIcons) Specifies icons to use for dark and light themes

Values for default_area:

- `maintoolbar`
- `formattoolbar`

Note: A manifest entry named `compose_action` is required to use `composeAction`.

7.2 Functions

7.2.1 setTitle(details)

Sets the title of the toolbar action. This shows up in the tooltip.

- `details` (object)
  - `title` (string or null) The string the toolbar action should display when moused over.

7.2.2 getTitle(details)

Gets the title of the toolbar action.

- `details` (*Details*)

Returns a Promise fulfilled with:

- string

7.2.3 setIcon(details)

Sets the icon for the toolbar action. The icon can be specified either as the path to an image file or as the pixel data from a canvas element, or as dictionary of either one of those. Either the **path** or the **imageData** property must be specified.

- `details` (object)
  - [`imageData`] (*ImageDataType* or object) Either an ImageData object or a dictionary {size -> ImageData} representing icon to be set. If the icon is specified as a dictionary, the actual image to be used is chosen depending on screen's pixel density. If the number of image pixels that fit into one screen space unit equals `scale`, then image with size `scale` * 19 will be selected. Initially only scales 1 and 2 will be supported. At least one image must be specified. Note that 'details.imageData = foo' is equivalent to 'details.imageData = {'19': foo}'
  - [`path`] (string or object) Either a relative image path or a dictionary {size -> relative image path} pointing to icon to be set. If the icon is specified as a dictionary, the actual image to be used is chosen depending on screen's pixel density. If the number of image pixels that fit into one screen space unit equals `scale`, then image with size `scale` * 19 will be selected. Initially only scales 1 and 2 will be supported. At least one image must be specified. Note that 'details.path = foo' is equivalent to 'details.imageData = {'19': foo}'

7.2.4 setPopup(details)

Sets the html document to be opened as a popup when the user clicks on the toolbar action's icon.

- `details` (object)
  - `popup` (string or null) The html file to show in a popup. If set to the empty string (''), no popup is shown.

7.2.5 getPopup(details)

Gets the html document set as the popup for this toolbar action.

- `details` (*Details*)

Returns a Promise fulfilled with:

- string

7.2.6 setBadgeText(details)

Sets the badge text for the toolbar action. The badge is displayed on top of the icon.

- `details` (object)
  - `text` (string or null) Any number of characters can be passed, but only about four can fit in the space.

7.2.7 getBadgeText(details)

Gets the badge text of the toolbar action. If no tab nor window is specified is specified, the global badge text is returned.

- `details` (*Details*)

Returns a Promise fulfilled with:

- string

7.2.8 setBadgeBackgroundColor(details)

Sets the background color for the badge.

- `details` (object)
  - `color` (string or *ColorArray* or null) An array of four integers in the range [0,255] that make up the RGBA color of the badge. For example, opaque red is `[255, 0, 0, 255]`. Can also be a string with a CSS value, with opaque red being `#FF0000` or `#F00`.

7.2.9 getBadgeBackgroundColor(details)

Gets the background color of the toolbar action.

- `details` (*Details*)

Returns a Promise fulfilled with:

- *ColorArray*

7.2.10 enable([tabId])

Enables the toolbar action for a tab. By default, toolbar actions are enabled.

- [`tabId`] (integer) The id of the tab for which you want to modify the toolbar action.

7.2.11 disable([tabId])

Disables the toolbar action for a tab.

- [`tabId`] (integer) The id of the tab for which you want to modify the toolbar action.

7.2.12 isEnabled(details)

Checks whether the toolbar action is enabled.

- `details` (*Details*)

7.2.13 openPopup()

Opens the extension popup window in the active window.

7.3 Events

7.3.1 onClicked()

Fired when a toolbar action icon is clicked. This event will not fire if the toolbar action has a popup.

7.4 Types

7.4.1 ColorArray

array of integer

7.4.2 Details

Specifies to which tab or window the value should be set, or from which one it should be retrieved. If no tab nor window is specified, the global value is set or retrieved.

object

- [`tabId`] (integer) When setting a value, it will be specific to the specified tab, and will automatically reset when the tab navigates. When getting, specifies the tab to get the value from; if there is no tab-specific value, the window one will be inherited.
- [`windowId`] (integer) When setting a value, it will be specific to the specified window. When getting, specifies the window to get the value from; if there is no window-specific value, the global one will be inherited.

7.4.3 ImageDataType

Pixel data for an image. Must be an ImageData object (for example, from a `canvas` element).

ImageData

CHAPTER 8

contacts

The address books API, also including the *addressBooks* and *mailingLists* namespaces, first appeared in Thunderbird 64. The quickSearch function was added in Thunderbird 68.

The Address Books sample extension uses this API.

Note: The permission `addressBooks` is required to use `contacts`.

8.1 Functions

8.1.1 list(parentId)

Gets all the contacts in the address book with the id `parentId`.

- `parentId` (string)

Returns a Promise fulfilled with:

- array of *ContactNode*

8.1.2 quickSearch([parentId], searchString)

Gets all contacts matching `searchString` in the address book with the id `parentId`.

- [`parentId`] (string) The ID of the address book to search. If not specified, all address books are searched.
- `searchString` (string) One or more space-separated terms to search for.

Returns a Promise fulfilled with:

- array of *ContactNode*

8.1.3 get(id)

Gets a single contact.

- `id` (string)

Returns a Promise fulfilled with:

- *ContactNode*

8.1.4 create(parentId, [id], properties)

Adds a new contact to the address book with the id `parentId`.

- `parentId` (string)
- [`id`] (string) Assigns the contact an id. If an existing contact has this id, an exception is thrown.
- `properties` (*ContactProperties*)

Returns a Promise fulfilled with:

- string The ID of the new contact.

8.1.5 update(id, properties)

Edits the properties of a contact. To remove a property, specify it as `null`.

- `id` (string)
- `properties` (*ContactProperties*)

8.1.6 delete(id)

Removes a contact from the address book. The contact is also removed from any mailing lists it is a member of.

- `id` (string)

8.2 Events

8.2.1 onCreated(node, id)

Fired when a contact is created.

- `node` (*ContactNode*)
- `id` (string)

8.2.2 onUpdated(node)

Fired when a contact is changed.

- `node` (*ContactNode*)

8.2.3 onDeleted(parentId, id)

Fired when a contact is removed from an address book.

- `parentId` (string)
- `id` (string)

8.3 Types

8.3.1 ContactNode

A node representing a contact in an address book.

object

- `id` (string) The unique identifier for the node. IDs are unique within the current profile, and they remain valid even after the program is restarted.
- `properties` (*ContactProperties*)
- `type` (*NodeType*) Always set to `contact`.
- [`parentId`] (string) The `id` of the parent object.
- [`readOnly`] (boolean) Indicates if the object is read-only. Currently this returns false in all cases, as read-only address books are ignored by the API.

8.3.2 ContactProperties

A set of properties for a particular contact. For a complete list of properties that Thunderbird uses, see https://hg.mozilla.org/comm-central/file/tip/mailnews/addrbook/public/nsIAbCard.idl

It is also possible to store custom properties. The custom property name however may only use a to z, A to Z, 1 to 9 and underscores.

object

CHAPTER 9

folders

The folders API first appeared in Thunderbird 68 (see bug 1531591) as a part of the *accounts* API. They were later moved here.

9.1 Permissions

- accountsFolders "Create, rename, or delete your mail account folders"

Note: The permission `accountsFolders` is required to use `folders`.

9.2 Functions

9.2.1 create(parentFolder, childName)

Creates a new subfolder of `parentFolder`.

- `parentFolder` (*MailFolder*)
- `childName` (string)

9.2.2 rename(folder, newName)

Renames a folder.

- `folder` (*MailFolder*)
- `newName` (string)

9.2.3 delete(folder)

Deletes a folder.

- `folder` (*MailFolder*)

9.3 Types

9.3.1 MailFolder

A folder object, as returned by the `list` and `get` methods. Use the accountId and path properties to refer to a folder.

object

- `accountId` (string) The account this folder belongs to.
- `path` (string) Path to this folder in the account. Although paths look predictable, never guess a folder's path, as there are a number of reasons why it may not be what you think it is.
- [`name`] (string) The human-friendly name of this folder.
- [`type`] (*string*) The type of folder, for several common types.

Values for type:

- `inbox`
- `drafts`
- `sent`
- `trash`
- `templates`
- `archives`
- `junk`
- `outbox`

CHAPTER 10

legacy

The legacy API first appeared in Thunderbird 63.

This API enables Thunderbird "legacy" extensions to continue working in the WebExtensions world. For much more information, see developer.thunderbird.net.

10.1 XUL Overlay Extensions

A new XUL overlay loader was created to replace the original one, which has been removed.

For the most part, things works exactly as they did before. However, as overlays are now added to a window's document *after* it has been parsed, you may experience some unusual behaviours. For example, your code may add a listener for an event that has already happened, or the UI doesn't return to its previous state correctly. Wibbly-wobbly, timey-wimey, ... stuff.

10.2 Bootstrapped Extensions

From Thunderbird 68, bootstrapped extensions are also required to use a WebExtensions-style `manifest.json`. This is the same as for an overlay extension, but with `type` set to `bootstrap`:

```
{
  ...
  "legacy": {
    "type": "bootstrap"
  }
  ...
}
```

(The only other possible value is `xul`, which is the default.)

10.3 How To Link Your Options Page

From Thunderbird 65, you can specify an options page in manifest.json, as you could in the old-style manifest. If you don't have an options page, just set `legacy` to `true`.

```
{
  ...
  "legacy": {
    "options": {
      "page": "chrome://address/of/your/options.page",
      "open_in_tab": true
    }
  }
  ...
}
```

10.4 Manifest file properties

- [`legacy`] (boolean or object)

CHAPTER 11

mailingLists

The address books API, also including the *addressBooks* and *contacts* namespaces, first appeared in Thunderbird 64.

The Address Books sample extension uses this API.

Note: The permission `addressBooks` is required to use `mailingLists`.

11.1 Functions

11.1.1 list(parentId)

Gets all the mailing lists in the address book with id `parentId`.

- `parentId` (string)

Returns a Promise fulfilled with:

- array of *MailingListNode*

11.1.2 get(id)

Gets a single mailing list.

- `id` (string)

Returns a Promise fulfilled with:

- *MailingListNode*

11.1.3 create(parentId, properties)

Creates a new mailing list in the address book with id `parentId`.

- `parentId` (string)
- `properties` (object)
  - `name` (string)
  - [`description`] (string)
  - [`nickName`] (string)

Returns a Promise fulfilled with:

- string The ID of the new mailing list.

11.1.4 update(id, properties)

Edits the properties of a mailing list.

- `id` (string)
- `properties` (object)
  - `name` (string)
  - [`description`] (string)
  - [`nickName`] (string)

11.1.5 delete(id)

Removes the mailing list.

- `id` (string)

11.1.6 addMember(id, contactId)

Adds a contact to the mailing list with id `id`. If the contact and mailing list are in different address books, the contact will also be copied to the list's address book.

- `id` (string)
- `contactId` (string)

11.1.7 listMembers(id)

Gets all contacts that are members of the mailing list with id `id`.

- `id` (string)

Returns a Promise fulfilled with:

- array of *ContactNode*

11.1.8 removeMember(id, contactId)

Removes a contact from the mailing list with id `id`. This does not delete the contact from the address book.

- `id` (string)
- `contactId` (string)

11.2 Events

11.2.1 onCreated(node)

Fired when a mailing list is created.

- `node` (*MailingListNode*)

11.2.2 onUpdated(node)

Fired when a mailing list is changed.

- `node` (*MailingListNode*)

11.2.3 onDeleted(parentId, id)

Fired when a mailing list is deleted.

- `parentId` (string)
- `id` (string)

11.2.4 onMemberAdded(node)

Fired when a contact is added to the mailing list.

- `node` (*ContactNode*)

11.2.5 onMemberRemoved(parentId, id)

Fired when a contact is removed from the mailing list.

- `parentId` (string)
- `id` (string)

11.3 Types

11.3.1 MailingListNode

A node representing a mailing list.

object

- `description` (string)
- `id` (string) The unique identifier for the node. IDs are unique within the current profile, and they remain valid even after the program is restarted.
- `name` (string)
- `nickName` (string)
- `type` (*NodeType*) Always set to `mailingList`.
- [`contacts`] (array of *ContactNode*) A list of contacts held by this node's address book or mailing list.
- [`parentId`] (string) The `id` of the parent object.
- [`readOnly`] (boolean) Indicates if the object is read-only. Currently this returns false in all cases, as read-only address books are ignored by the API.

CHAPTER 12

mailTabs

The messages API first appeared in Thunderbird 66 (see bug 1499617).

The Filter and Layout sample extensions use this API.

12.1 Functions

12.1.1 query(queryInfo)

Gets all mail tabs that have the specified properties, or all mail tabs if no properties are specified.

- `queryInfo` (object)
  - [`active`] (boolean) Whether the tabs are active in their windows.
  - [`currentWindow`] (boolean) Whether the tabs are in the current window.
  - [`lastFocusedWindow`] (boolean) Whether the tabs are in the last focused window.
  - [`windowId`] (integer) The ID of the parent window, or *WINDOW_ID_CURRENT* for the current window.

Returns a Promise fulfilled with:

- array of *MailTab*

12.1.2 update([tabId], updateProperties)

Modifies the properties of a mail tab. Properties that are not specified in `updateProperties` are not modified.

- [`tabId`] (integer) Defaults to the active tab of the current window.
- `updateProperties` (object)
  - [`displayedFolder`] (*MailFolder*) Sets the folder displayed in the tab. The extension must have an accounts permission to do this.

- [folderPaneVisible] (boolean) Shows or hides the folder pane.
- [layout] (*string*) Sets the arrangement of the folder pane, message list pane, and message display pane. Note that setting this applies it to all mail tabs.
- [messagePaneVisible] (boolean) Shows or hides the message display pane.
- [sortOrder] (*string*) Sorts the list of messages. sortType must also be given.
- [sortType] (*string*) Sorts the list of messages. sortOrder must also be given.

Values for layout:

- standard
- wide
- vertical

Values for sortOrder:

- none
- ascending
- descending

Values for sortType:

- none
- date
- subject
- author
- id
- thread
- priority
- status
- size
- flagged
- unread
- recipient
- location
- tags
- junkStatus
- attachments
- account
- custom
- received
- correspondent

12.1.3 getSelectedMessages([tabId])

Lists the selected messages in the current folder. A messages permission is required to do this.

- [`tabId`] (integer) Defaults to the active tab of the current window.

Returns a Promise fulfilled with:

- *MessageList*

12.1.4 setQuickFilter([tabId], properties)

Sets the Quick Filter user interface based on the options specified.

- [`tabId`] (integer) Defaults to the active tab of the current window.
- `properties` (object)
  - [`attachment`] (boolean) Shows only messages with attachments.
  - [`contact`] (boolean) Shows only messages from people in the address book.
  - [`flagged`] (boolean) Shows only flagged messages.
  - [`show`] (boolean) Shows or hides the Quick Filter bar.
  - [`tags`] (boolean or *QuickFilterTagsDetail*) Shows only messages with tags on them.
  - [`text`] (*QuickFilterTextDetail*) Shows only messages matching the supplied text.
  - [`unread`] (boolean) Shows only unread messages.

12.2 Events

12.2.1 onDisplayedFolderChanged()

Fired when the displayed folder changes in any mail tab.

Note: The permission `accountsRead` is required to use `onDisplayedFolderChanged`.

12.2.2 onSelectedMessagesChanged()

Fired when the selected messages change in any mail tab.

Note: The permission `messagesRead` is required to use `onSelectedMessagesChanged`.

12.3 Types

12.3.1 MailTab

object

- `active` (boolean)
- `displayedFolder` (*MailFolder*)
- `folderPaneVisible` (boolean)
- `id` (integer)
- `layout` (*string*)
- `messagePaneVisible` (boolean)
- `sortOrder` (*string*)
- `sortType` (*string*)
- `windowId` (integer)

Values for layout:

- `standard`
- `wide`
- `vertical`

Values for sortOrder:

- `none`
- `ascending`
- `descending`

Values for sortType:

- `none`
- `date`
- `subject`
- `author`
- `id`
- `thread`
- `priority`
- `status`
- `size`
- `flagged`
- `unread`
- `recipient`
- `location`
- `tags`
- `junkStatus`
- `attachments`
- `account`
- `custom`

- `received`
- `correspondent`

12.3.2 QuickFilterTagsDetail

object

- `mode` (*string*) Whether all of the tag filters must apply, or any of them.
- `tags` (object) Object keys are tags to filter on, values are `true` if the message must have the tag, or `false` if it must not have the tag. For a list of available tags, call the *listTags()* method.

Values for mode:

- `all`
- `any`

12.3.3 QuickFilterTextDetail

object

- `text` (string) String to match against the `recipients`, `author`, `subject`, or `body`.
- [`author`] (boolean) Shows messages where `text` matches the author.
- [`body`] (boolean) Shows messages where `text` matches the message body.
- [`recipients`] (boolean) Shows messages where `text` matches the recipients.
- [`subject`] (boolean) Shows messages where `text` matches the subject.

CHAPTER 13

menus

The menus API first appeared in Thunderbird 66 (see bug 1503421). It is basically the same as the Firefox menus API, but modified to suit Thunderbird. Note that the similar `contextMenus` API will not be added to Thunderbird.

Use the browser.menus API to add items to the browser's menus. You can choose what types of objects your context menu additions apply to, such as images, hyperlinks, and pages.

13.1 Permissions

- menus
- menus.overrideContext

Note: The permission `menus` is required to use `menus`.

13.2 Functions

13.2.1 create(createProperties, [callback])

Creates a new context menu item. Note that if an error occurs during creation, you may not find out until the creation callback fires (the details will be in runtime.lastError).

- `createProperties` (object)
  - [`checked`] (boolean) The initial state of a checkbox or radio item: true for selected and false for unselected. Only one radio item can be selected at a time in a given group of radio items.
  - [`command`] (string) Specifies a command to issue for the context click. Currently supports internal command _execute_browser_action.

  - [contexts] (array of *ContextType*) List of contexts this menu item will appear in. Defaults to ['page'] if not specified.
  - [documentUrlPatterns] (array of string) Lets you restrict the item to apply only to documents whose URL matches one of the given patterns. (This applies to frames as well.) For details on the format of a pattern, see Match Patterns.
  - [enabled] (boolean) Whether this context menu item is enabled or disabled. Defaults to true.
  - [icons] (object)
  - [id] (string) The unique ID to assign to this item. Mandatory for event pages. Cannot be the same as another ID for this extension.
  - [onclick] (function) A function that will be called back when the menu item is clicked. Event pages cannot use this.
  - [parentId] (integer or string) The ID of a parent menu item; this makes the item a child of a previously added item.
  - [targetUrlPatterns] (array of string) Similar to documentUrlPatterns, but lets you filter based on the src attribute of img/audio/video tags and the href of anchor tags.
  - [title] (string) The text to be displayed in the item; this is *required* unless type is 'separator'. When the context is 'selection', you can use %s within the string to show the selected text. For example, if this parameter's value is "Translate '%s' to Pig Latin" and the user selects the word "cool", the context menu item for the selection is "Translate 'cool' to Pig Latin".
  - [type] (*ItemType*) The type of menu item. Defaults to 'normal' if not specified.
  - [viewTypes] (array of ViewType) List of view types where the menu item will be shown. Defaults to any view, including those without a viewType.
  - [visible] (boolean) Whether the item is visible in the menu.
- [callback] (function) Called when the item has been created in the browser. If there were any problems creating the item, details will be available in runtime.lastError.

Returns a Promise fulfilled with:

- integer or string The ID of the newly created item.

13.2.2 update(id, updateProperties)

Updates a previously created context menu item.

- id (integer or string) The ID of the item to update.
- updateProperties (object) The properties to update. Accepts the same values as the create function.
  - [checked] (boolean)
  - [contexts] (array of *ContextType*)
  - [documentUrlPatterns] (array of string)
  - [enabled] (boolean)
  - [icons] (object)
  - [onclick] (function)
  - [parentId] (integer or string) Note: You cannot change an item to be a child of one of its own descendants.

  - [targetUrlPatterns] (array of string)
  - [title] (string)
  - [type] (*ItemType*)
  - [viewTypes] (array of ViewType)
  - [visible] (boolean) Whether the item is visible in the menu.

13.2.3 remove(menuItemId)

Removes a context menu item.

- menuItemId (integer or string) The ID of the context menu item to remove.

13.2.4 removeAll()

Removes all context menu items added by this extension.

13.2.5 overrideContext(contextOptions)

Show the matching menu items from this extension instead of the default menu. This should be called during a 'contextmenu' DOM event handler, and only applies to the menu that opens after this event.

- contextOptions (object)
  - [context] (*string*) ContextType to override, to allow menu items from other extensions in the menu. Currently only 'tab' is supported. showDefaults cannot be used with this option.
  - [showDefaults] (boolean) Whether to also include default menu items in the menu.
  - [tabId] (integer) Required when context is 'tab'. Requires 'tabs' permission.

Values for context:

- tab

Note: The permission menus.overrideContext is required to use overrideContext.

13.2.6 refresh()

Updates the extension items in the shown menu, including changes that have been made since the menu was shown. Has no effect if the menu is hidden. Rebuilding a shown menu is an expensive operation, only invoke this method when necessary.

13.3 Events

13.3.1 onClicked(info, [tab])

Fired when a context menu item is clicked.

- info (*OnClickData*) Information about the item clicked and the context where the click happened.

- [tab] (*Tab*) The details of the tab where the click took place. If the click did not take place in a tab, this parameter will be missing.

13.3.2 onShown(info, tab)

Fired when a menu is shown. The extension can add, modify or remove menu items and call menus.refresh() to update the menu.

- info (object) Information about the context of the menu action and the created menu items. For more information about each property, see OnClickData. The following properties are only set if the extension has host permissions for the given context: linkUrl, linkText, srcUrl, pageUrl, frameUrl, selectionText.
  - contexts (array of *ContextType*) A list of all contexts that apply to the menu.
  - editable (boolean)
  - menuIds (array of None) A list of IDs of the menu items that were shown.
  - [frameUrl] (string)
  - [linkText] (string)
  - [linkUrl] (string)
  - [mediaType] (string)
  - [pageUrl] (string)
  - [selectionText] (string)
  - [srcUrl] (string)
  - [targetElementId] (integer)
  - [viewType] (ViewType)
- tab (*Tab*) The details of the tab where the menu was opened.

13.3.3 onHidden()

Fired when a menu is hidden. This event is only fired if onShown has fired before.

13.4 Properties

13.4.1 ACTION_MENU_TOP_LEVEL_LIMIT

The maximum number of top level extension items that can be added to an extension action context menu. Any items beyond this limit will be ignored.

13.5 Types

13.5.1 ContextType

The different contexts a menu can appear in. Specifying 'all' is equivalent to the combination of all other contexts except for 'tab'.

string

Values for ContextType:

- `all`
- `page`
- `frame`
- `selection`
- `link`
- `editable`
- `password`
- `image`
- `video`
- `audio`
- `browser_action`
- `tab`
- `message_list`
- `folder_pane`

13.5.2 ItemType

The type of menu item.

string

Values for ItemType:

- `normal`
- `checkbox`
- `radio`
- `separator`

13.5.3 OnClickData

Information sent when a context menu item is clicked.

object

- `editable` (boolean) A flag indicating whether the element is editable (text input, textarea, etc.).
- `menuItemId` (integer or string) The ID of the menu item that was clicked.
- `modifiers` (array of *string*) An array of keyboard modifiers that were held while the menu item was clicked.
- [`button`] (integer) An integer value of button by which menu item was clicked.
- [`checked`] (boolean) A flag indicating the state of a checkbox or radio item after it is clicked.
- [`displayedFolder`] (*MailFolder*) The displayed folder, if the context menu was opened in the message list.
- [`frameId`] (integer) The id of the frame of the element where the context menu was clicked.

- [frameUrl] (string) The URL of the frame of the element where the context menu was clicked, if it was in a frame.
- [linkText] (string) If the element is a link, the text of that link.
- [linkUrl] (string) If the element is a link, the URL it points to.
- [mediaType] (string) One of 'image', 'video', or 'audio' if the context menu was activated on one of these types of elements.
- [pageUrl] (string) The URL of the page where the menu item was clicked. This property is not set if the click occurred in a context where there is no current page, such as in a launcher context menu.
- [parentMenuItemId] (integer or string) The parent ID, if any, for the item clicked.
- [selectedFolder] (*MailFolder*) The selected folder, if the context menu was opened in the folder pane.
- [selectedMessages] (*MessageList*) The selected messages, if the context menu was opened in the message list.
- [selectionText] (string) The text for the context selection, if any.
- [srcUrl] (string) Will be present for elements with a 'src' URL.
- [targetElementId] (integer) An identifier of the clicked element, if any. Use menus.getTargetElement in the page to find the corresponding element.
- [viewType] (ViewType) The type of view where the menu is clicked. May be unset if the menu is not associated with a view.
- [wasChecked] (boolean) A flag indicating the state of a checkbox or radio item before it was clicked.

Values for modifiers:

- Shift
- Alt
- Command
- Ctrl
- MacCtrl

CHAPTER 14

messageDisplay

The message display API first appeared in Thunderbird 70 and was backported to Thunderbird 68.2.

A message can be displayed in either a 3-pane tab, a tab of its own, or in a window of its own. All are referenced by `tabId` in this API. Display windows are considered to have exactly one tab, which has limited functionality compared to tabs from the main window.

More functions are planned for this API for adding to the user interface, as well as a message display action (similar to *browserAction* and *composeAction*).

Note: The permission `messagesRead` is required to use `messageDisplay`.

14.1 Functions

14.1.1 getDisplayedMessage(tabId)

Gets the currently displayed message in the specified tab, or null if no message is displayed.

- `tabId` (integer)

Returns a Promise fulfilled with:

- *MessageHeader*

14.2 Events

14.2.1 onMessageDisplayed(tabId, message)

Fired when a message is displayed, whether in a 3-pane tab, a message tab, or a message window.

- `tabId` (integer)

- `message` (*MessageHeader*)

CHAPTER 15

messageDisplayAction

The messageDisplayAction API was added in Thunderbird 71, and was uplifted to Thunderbird 68.3 ESR. It is similar to Firefox's browserAction API and can be combined with the *messageDisplay* API to determine the currently displayed message.

Use toolbar actions to put icons in the mail window toolbar. In addition to its icon, a toolbar action can also have a tooltip, a badge, and a popup. This namespace is called browserAction for compatibility with browser WebExtensions.

15.1 Manifest file properties

- [`message_display_action`] (object)
  - [`browser_style`] (boolean)
  - [`default_area`] (string) Currently unused.
  - [`default_icon`] (IconPath)
  - [`default_popup`] (string)
  - [`default_title`] (string)
  - [`theme_icons`] (array of ThemeIcons) Specifies icons to use for dark and light themes

Note: A manifest entry named `message_display_action` is required to use `messageDisplayAction`.

15.2 Functions

15.2.1 setTitle(details)

Sets the title of the toolbar action. This shows up in the tooltip.

- `details` (object)
  - `title` (string or null) The string the toolbar action should display when moused over.

15.2.2 getTitle(details)

Gets the title of the toolbar action.

- `details` (*Details*)

Returns a Promise fulfilled with:

- string

15.2.3 setIcon(details)

Sets the icon for the toolbar action. The icon can be specified either as the path to an image file or as the pixel data from a canvas element, or as dictionary of either one of those. Either the **path** or the **imageData** property must be specified.

- `details` (object)
  - [`imageData`] (*ImageDataType* or object) Either an ImageData object or a dictionary {size -> ImageData} representing icon to be set. If the icon is specified as a dictionary, the actual image to be used is chosen depending on screen's pixel density. If the number of image pixels that fit into one screen space unit equals `scale`, then image with size `scale` * 19 will be selected. Initially only scales 1 and 2 will be supported. At least one image must be specified. Note that 'details.imageData = foo' is equivalent to 'details.imageData = {'19': foo}'
  - [`path`] (string or object) Either a relative image path or a dictionary {size -> relative image path} pointing to icon to be set. If the icon is specified as a dictionary, the actual image to be used is chosen depending on screen's pixel density. If the number of image pixels that fit into one screen space unit equals `scale`, then image with size `scale` * 19 will be selected. Initially only scales 1 and 2 will be supported. At least one image must be specified. Note that 'details.path = foo' is equivalent to 'details.imageData = {'19': foo}'

15.2.4 setPopup(details)

Sets the html document to be opened as a popup when the user clicks on the toolbar action's icon.

- `details` (object)
  - `popup` (string or null) The html file to show in a popup. If set to the empty string (''), no popup is shown.

15.2.5 getPopup(details)

Gets the html document set as the popup for this toolbar action.

- `details` (*Details*)

Returns a Promise fulfilled with:

- string

15.2.6 setBadgeText(details)

Sets the badge text for the toolbar action. The badge is displayed on top of the icon.

- `details` (object)
  - `text` (string or null) Any number of characters can be passed, but only about four can fit in the space.

15.2.7 getBadgeText(details)

Gets the badge text of the toolbar action. If no tab nor window is specified is specified, the global badge text is returned.

- `details` (*Details*)

Returns a Promise fulfilled with:

- string

15.2.8 setBadgeBackgroundColor(details)

Sets the background color for the badge.

- `details` (object)
  - `color` (string or *ColorArray* or null) An array of four integers in the range [0,255] that make up the RGBA color of the badge. For example, opaque red is `[255, 0, 0, 255]`. Can also be a string with a CSS value, with opaque red being `#FF0000` or `#F00`.

15.2.9 getBadgeBackgroundColor(details)

Gets the background color of the toolbar action.

- `details` (*Details*)

Returns a Promise fulfilled with:

- *ColorArray*

15.2.10 enable([tabId])

Enables the toolbar action for a tab. By default, toolbar actions are enabled.

- [`tabId`] (integer) The id of the tab for which you want to modify the toolbar action.

15.2.11 disable([tabId])

Disables the toolbar action for a tab.

- [`tabId`] (integer) The id of the tab for which you want to modify the toolbar action.

15.2.12 isEnabled(details)

Checks whether the toolbar action is enabled.

- `details` (*Details*)

15.2.13 openPopup()

Opens the extension popup window in the active window.

15.3 Events

15.3.1 onClicked(tabId)

Fired when a toolbar action icon is clicked. This event will not fire if the toolbar action has a popup.

- `tabId` (integer)

15.4 Types

15.4.1 ColorArray

array of integer

15.4.2 Details

Specifies to which tab or window the value should be set, or from which one it should be retrieved. If no tab nor window is specified, the global value is set or retrieved.

object

- [`tabId`] (integer) When setting a value, it will be specific to the specified tab, and will automatically reset when the tab navigates. When getting, specifies the tab to get the value from; if there is no tab-specific value, the window one will be inherited.
- [`windowId`] (integer) When setting a value, it will be specific to the specified window. When getting, specifies the window to get the value from; if there is no window-specific value, the global one will be inherited.

15.4.3 ImageDataType

Pixel data for an image. Must be an ImageData object (for example, from a `canvas` element).

ImageData

CHAPTER 16

messages

The messages API first appeared in Thunderbird 66.

Note: When the term `messageId` is used in these documents, it *doesn't* refer to the Message-ID email header. It is an internal tracking number that does not remain after a restart. Nor does it follow an email that has been moved to a different folder.

Warning: Some functions in this API potentially return *a lot* of messages. Be careful what you wish for! See *Working with Message Lists* for more information.

16.1 Permissions

- messagesMove "Move, copy, or delete your email messages"
- messagesRead "Read your email messages and mark or tag them"

Note: The permission `messagesRead` is required to use `messages`.

16.2 Functions

16.2.1 list(folder)

Gets all messages in a folder.

- `folder` (*MailFolder*)

Returns a Promise fulfilled with:

- *MessageList*

16.2.2 continueList(messageListId)

Returns the next chunk of messages in a list. See *Working with Message Lists* for more information.

- `messageListId` (string)

Returns a Promise fulfilled with:

- *MessageList*

16.2.3 get(messageId)

Returns a specified message.

- `messageId` (integer)

Returns a Promise fulfilled with:

- *MessageHeader*

16.2.4 getFull(messageId)

Returns a specified message, including all headers and MIME parts.

- `messageId` (integer)

Returns a Promise fulfilled with:

- *MessagePart*

16.2.5 getRaw(messageId)

Added in Thunderbird 72

Returns the unmodified source of a message.

- `messageId` (integer)

Returns a Promise fulfilled with:

- *MessageList*

16.2.6 query(queryInfo)

Added in Thunderbird 69

Backported to Thunderbird 68.2

Gets all messages that have the specified properties, or all messages if no properties are specified.

- `queryInfo` (object)
  - [`author`] (string) Returns only messages with this value matching the author.
  - [`body`] (string) Returns only messages with this value in the body of the mail.
  - [`flagged`] (boolean) Returns only flagged (or unflagged if false) messages.

  - [folder] (*MailFolder*) Returns only messages from the specified folder.
  - [fromDate] (Date) Returns only messages with a date after this value.
  - [fromMe] (boolean) Returns only messages with the author matching any configured identity.
  - [fullText] (string) Returns only messages with this value somewhere in the mail (subject, body or author).
  - [recipients] (string) Returns only messages with this value matching one or more recipients.
  - [subject] (string) Returns only messages with this value matching the subject.
  - [toDate] (Date) Returns only messages with a date before this value.
  - [toMe] (boolean) Returns only messages with one or more recipients matching any configured identity.
  - [unread] (boolean) Returns only unread (or read if false) messages.

Returns a Promise fulfilled with:

- *MessageList*

16.2.7 update(messageId, newProperties)

Marks or unmarks a message as read, flagged, or tagged.

- messageId (integer)
- newProperties (object)
  - [flagged] (boolean) Marks the message as flagged or unflagged.
  - [junk] (boolean) Marks the message as junk or not junk. *Added in Thunderbird 73*
  - [read] (boolean) Marks the message as read or unread.
  - [tags] (array of string) Sets the tags on the message. For a list of available tags, call the listTags method.

16.2.8 move(messageIds, destination)

Moves messages to a specified folder.

- messageIds (array of integer) The IDs of the messages to move.
- destination (*MailFolder*) The folder to move the messages to.

Note: The permission messagesMove is required to use move.

16.2.9 copy(messageIds, destination)

Copies messages to a specified folder.

- messageIds (array of integer) The IDs of the messages to copy.
- destination (*MailFolder*) The folder to copy the messages to.

Note: The permission messagesMove is required to use copy.

16.2.10 delete(messageIds, [skipTrash])

Deletes messages, or moves them to the trash folder.

- `messageIds` (array of integer) The IDs of the messages to delete.
- [`skipTrash`] (boolean) If true, the message will be permanently deleted without warning the user. If false or not specified, it will be moved to the trash folder.

Note: The permission `messagesMove` is required to use `delete`.

16.2.11 archive(messageIds)

Archives messages using the current settings.

- `messageIds` (array of integer) The IDs of the messages to archive.

Note: The permission `messagesMove` is required to use `archive`.

16.2.12 listTags()

Returns a list of tags that can be set on messages, and their human-friendly name, colour, and sort order.

Returns a Promise fulfilled with:

- array of *MessageTag*

16.3 Types

16.3.1 MessageHeader

object

- `author` (string)
- `bccList` (array of string)
- `ccList` (array of string)
- `date` (date)
- `flagged` (boolean)
- `folder` (*MailFolder*)
- `id` (integer)
- `read` (boolean)
- `recipients` (array of string)
- `subject` (string)
- `tags` (array of string)

16.3.2 MessageList

See *Working with Message Lists* for more information.

object

- `id` (string)
- `messages` (array of *MessageHeader*)

16.3.3 MessagePart

Represents an email message "part", which could be the whole message

object

- [`body`] (string) The content of the part
- [`contentType`] (string)
- [`headers`] (object) An object of part headers, with the header name as key, and an array of header values as value
- [`name`] (string) Name of the part, if it is a file
- [`partName`] (string)
- [`parts`] (array of *MessagePart*) Any sub-parts of this part
- [`size`] (integer)

16.3.4 MessageTag

object

- `color` (string) Tag color
- `key` (string) Distinct tag identifier – use this string when referring to a tag
- `ordinal` (string) Custom sort string (usually empty)
- `tag` (string) Human-readable tag name

CHAPTER 17

tabs

Use the `browser.tabs` API to interact with the browser's tab system. You can use this API to create, modify, and rearrange tabs in the browser.

17.1 Permissions

- activeTab
- tabs "Access browser tabs"
- tabHide "Hide and show browser tabs"

17.2 Functions

17.2.1 get(tabId)

Retrieves details about the specified tab.

- `tabId` (integer)

Returns a Promise fulfilled with:

- *Tab*

17.2.2 getCurrent()

Gets the tab that this script call is being made from. May be undefined if called from a non-tab context (for example: a background page or popup view).

Returns a Promise fulfilled with:

- *Tab*

17.2.3 create(createProperties)

Creates a new tab.

- `createProperties` (object)
  - [`active`] (boolean) Whether the tab should become the active tab in the window. Does not affect whether the window is focused (see *update(windowId, updateInfo)*). Defaults to `true`.
  - [`index`] (integer) The position the tab should take in the window. The provided value will be clamped to between zero and the number of tabs in the window.
  - [`selected`] (boolean) **Deprecated.** Whether the tab should become the selected tab in the window. Defaults to `true`
  - [`url`] (string) The URL to navigate the tab to initially. Fully-qualified URLs must include a scheme (i.e. 'http://www.google.com', not 'www.google.com'). Relative URLs will be relative to the current page within the extension. Defaults to the New Tab Page.
  - [`windowId`] (integer) The window to create the new tab in. Defaults to the current window.

Returns a Promise fulfilled with:

- *Tab* Details about the created tab. Will contain the ID of the new tab.

17.2.4 duplicate(tabId)

Duplicates a tab.

- `tabId` (integer) The ID of the tab which is to be duplicated.

Returns a Promise fulfilled with:

- *Tab* Details about the duplicated tab. The *Tab* object doesn't contain `url`, `title` and `favIconUrl` if the `"tabs"` permission has not been requested.

17.2.5 query(queryInfo)

Gets all tabs that have the specified properties, or all tabs if no properties are specified.

- `queryInfo` (object)
  - [`active`] (boolean) Whether the tabs are active in their windows.
  - [`currentWindow`] (boolean) Whether the tabs are in the current window.
  - [`highlighted`] (boolean) Whether the tabs are highlighted. Works as an alias of active.
  - [`index`] (integer) The position of the tabs within their windows.
  - [`lastFocusedWindow`] (boolean) Whether the tabs are in the last focused window.
  - [`mailTab`] (boolean) Whether the tab is a Thunderbird 3-pane tab.
  - [`status`] (*TabStatus*) Whether the tabs have completed loading.
  - [`title`] (string) Match page titles against a pattern.
  - [`url`] (string or array of string) Match tabs against one or more URL Patterns. Note that fragment identifiers are not matched.
  - [`windowId`] (integer) The ID of the parent window, or *WINDOW_ID_CURRENT* for the current window.
  - [`windowType`] (*WindowType*) The type of window the tabs are in.

Returns a Promise fulfilled with:

- array of *Tab*

17.2.6 update([tabId], updateProperties)

Modifies the properties of a tab. Properties that are not specified in `updateProperties` are not modified.

- [`tabId`] (integer) Defaults to the selected tab of the current window.
- `updateProperties` (object)
  - [`active`] (boolean) Whether the tab should be active. Does not affect whether the window is focused (see *update(windowId, updateInfo)*).
  - [`url`] (string) A URL to navigate the tab to.

Returns a Promise fulfilled with:

- *Tab* Details about the updated tab. The *Tab* object doesn't contain `url`, `title` and `favIconUrl` if the `"tabs"` permission has not been requested.

17.2.7 move(tabIds, moveProperties)

Moves one or more tabs to a new position within its window, or to a new window. Note that tabs can only be moved to and from normal (window.type === "normal") windows.

- `tabIds` (integer or array of integer) The tab or list of tabs to move.
- `moveProperties` (object)
  - `index` (integer) The position to move the window to. -1 will place the tab at the end of the window.
  - [`windowId`] (integer) Defaults to the window the tab is currently in.

Returns a Promise fulfilled with:

- *Tab* or array of *Tab* Details about the moved tabs.

17.2.8 reload([tabId], [reloadProperties])

Reload a tab.

- [`tabId`] (integer) The ID of the tab to reload; defaults to the selected tab of the current window.
- [`reloadProperties`] (object)
  - [`bypassCache`] (boolean) Whether using any local cache. Default is false.

17.2.9 remove(tabIds)

Closes one or more tabs.

- `tabIds` (integer or array of integer) The tab or list of tabs to close.

17.2.10 executeScript([tabId], details)

Injects JavaScript code into a page. For details, see the programmatic injection section of the content scripts doc.

- [`tabId`] (integer) The ID of the tab in which to run the script; defaults to the active tab of the current window.
- `details` (InjectDetails) Details of the script to run.

Returns a Promise fulfilled with:

- array of any The result of the script in every injected frame.

17.2.11 insertCSS([tabId], details)

Injects CSS into a page. For details, see the programmatic injection section of the content scripts doc.

- [`tabId`] (integer) The ID of the tab in which to insert the CSS; defaults to the active tab of the current window.
- `details` (InjectDetails) Details of the CSS text to insert.

17.2.12 removeCSS([tabId], details)

Removes injected CSS from a page. For details, see the programmatic injection section of the content scripts doc.

- [`tabId`] (integer) The ID of the tab from which to remove the injected CSS; defaults to the active tab of the current window.
- `details` (InjectDetails) Details of the CSS text to remove.

17.3 Events

17.3.1 onCreated(tab)

Fired when a tab is created. Note that the tab's URL may not be set at the time this event fired, but you can listen to onUpdated events to be notified when a URL is set.

- `tab` (*Tab*) Details of the tab that was created.

17.3.2 onUpdated(tabId, changeInfo, tab)

Fired when a tab is updated.

- `tabId` (integer)
- `changeInfo` (object) Lists the changes to the state of the tab that was updated.
  - [`favIconUrl`] (string) The tab's new favicon URL.
  - [`status`] (string) The status of the tab. Can be either *loading* or *complete*.
  - [`url`] (string) The tab's URL if it has changed.
- `tab` (*Tab*) Gives the state of the tab that was updated.

17.3.3 onMoved(tabId, moveInfo)

Fired when a tab is moved within a window. Only one move event is fired, representing the tab the user directly moved. Move events are not fired for the other tabs that must move in response. This event is not fired when a tab is moved between windows. For that, see *onDetached(tabId, detachInfo)*.

- `tabId` (integer)
- `moveInfo` (object)
  - `fromIndex` (integer)
  - `toIndex` (integer)
  - `windowId` (integer)

17.3.4 onActivated(activeInfo)

Fires when the active tab in a window changes. Note that the tab's URL may not be set at the time this event fired, but you can listen to onUpdated events to be notified when a URL is set.

- `activeInfo` (object)
  - `tabId` (integer) The ID of the tab that has become active.
  - `windowId` (integer) The ID of the window the active tab changed inside of.

17.3.5 onDetached(tabId, detachInfo)

Fired when a tab is detached from a window, for example because it is being moved between windows.

- `tabId` (integer)
- `detachInfo` (object)
  - `oldPosition` (integer)
  - `oldWindowId` (integer)

17.3.6 onAttached(tabId, attachInfo)

Fired when a tab is attached to a window, for example because it was moved between windows.

- `tabId` (integer)
- `attachInfo` (object)
  - `newPosition` (integer)
  - `newWindowId` (integer)

17.3.7 onRemoved(tabId, removeInfo)

Fired when a tab is closed.

- `tabId` (integer)
- `removeInfo` (object)
  - `isWindowClosing` (boolean) True when the tab is being closed because its window is being closed.

- `windowId` (integer) The window whose tab is closed.

17.4 Properties

17.4.1 TAB_ID_NONE

An ID which represents the absence of a browser tab.

17.5 Types

17.5.1 Tab

object

- `active` (boolean) Whether the tab is active in its window. (Does not necessarily mean the window is focused.)
- `highlighted` (boolean) Whether the tab is highlighted. Works as an alias of active
- `index` (integer) The zero-based index of the tab within its window.
- `selected` (boolean) **Deprecated.** Whether the tab is selected.
- [`favIconUrl`] (string) The URL of the tab's favicon. This property is only present if the extension's manifest includes the `"tabs"` permission. It may also be an empty string if the tab is loading.
- [`height`] (integer) The height of the tab in pixels.
- [`id`] (integer) The ID of the tab. Tab IDs are unique within a browser session. Under some circumstances a Tab may not be assigned an ID. Tab ID can also be set to *TAB_ID_NONE* for apps and devtools windows.
- [`mailTab`] (boolean) Whether the tab is a 3-pane tab.
- [`status`] (string) Either *loading* or *complete*.
- [`title`] (string) The title of the tab. This property is only present if the extension's manifest includes the `"tabs"` permission.
- [`url`] (string) The URL the tab is displaying. This property is only present if the extension's manifest includes the `"tabs"` permission.
- [`width`] (integer) The width of the tab in pixels.
- [`windowId`] (integer) The ID of the window the tab is contained within.

17.5.2 TabStatus

Whether the tabs have completed loading.

string

Values for TabStatus:

- `loading`
- `complete`

17.5.3 UpdateFilter

An object describing filters to apply to tabs.onUpdated events.

object

- [`properties`] (array of *UpdatePropertyName*) A list of property names. Events that do not match any of the names will be filtered out.
- [`tabId`] (integer)
- [`urls`] (array of string) A list of URLs or URL patterns. Events that cannot match any of the URLs will be filtered out. Filtering with urls requires the `"tabs"` or `"activeTab"` permission.
- [`windowId`] (integer)

17.5.4 UpdatePropertyName

Event names supported in onUpdated.

string

Values for UpdatePropertyName:

- `favIconUrl`
- `status`
- `title`

17.5.5 WindowType

The type of window.

string

Values for WindowType:

- `normal`
- `popup`
- `panel`
- `app`
- `devtools`

CHAPTER 18

windows

Note: These APIs are for the main Thunderbird windows which can contain webpage tabs and also other window types like composer or address books that cannot contain webpage tabs. Make sure your code interacts with windows appropriately, depending on their type.

Use the `browser.windows` API to interact with Thunderbird. You can use this API to create, modify, and rearrange windows.

18.1 Functions

18.1.1 get(windowId, [getInfo])

Gets details about a window.

- `windowId` (integer)
- [`getInfo`] (object)
  - [`populate`] (boolean) If true, the *Window* object will have a `tabs` property that contains a list of the *Tab* objects. The `Tab` objects only contain the `url`, `title` and `favIconUrl` properties if the extension's manifest file includes the `"tabs"` permission.
  - [`windowTypes`] (array of *WindowType*) If set, the *Window* returned will be filtered based on its type. If unset the default filter is set to `['app', 'normal', 'panel', 'popup']`, with `'app'` and `'panel'` window types limited to the extension's own windows.

Returns a Promise fulfilled with:

- *Window*

18.1.2 getCurrent([getInfo])

Gets the current window.

- [getInfo] (object)
  - [populate] (boolean) If true, the *Window* object will have a tabs property that contains a list of the *Tab* objects. The Tab objects only contain the url, title and favIconUrl properties if the extension's manifest file includes the "tabs" permission.
  - [windowTypes] (array of *WindowType*) If set, the *Window* returned will be filtered based on its type. If unset the default filter is set to ['app', 'normal', 'panel', 'popup'], with 'app' and 'panel' window types limited to the extension's own windows.

Returns a Promise fulfilled with:

- *Window*

18.1.3 getLastFocused([getInfo])

Gets the window that was most recently focused — typically the window 'on top'.

- [getInfo] (object)
  - [populate] (boolean) If true, the *Window* object will have a tabs property that contains a list of the *Tab* objects. The Tab objects only contain the url, title and favIconUrl properties if the extension's manifest file includes the "tabs" permission.
  - [windowTypes] (array of *WindowType*) If set, the *Window* returned will be filtered based on its type. If unset the default filter is set to ['app', 'normal', 'panel', 'popup'], with 'app' and 'panel' window types limited to the extension's own windows.

Returns a Promise fulfilled with:

- *Window*

18.1.4 getAll([getInfo])

Gets all windows.

- [getInfo] (object)
  - [populate] (boolean) If true, each *Window* object will have a tabs property that contains a list of the *Tab* objects for that window. The Tab objects only contain the url, title and favIconUrl properties if the extension's manifest file includes the "tabs" permission.
  - [windowTypes] (array of *WindowType*) If set, the *Window* returned will be filtered based on its type. If unset the default filter is set to ['app', 'normal', 'panel', 'popup'], with 'app' and 'panel' window types limited to the extension's own windows.

Returns a Promise fulfilled with:

- array of *Window*

18.1.5 create([createData])

Creates (opens) a new browser with any optional sizing, position or default URL provided.

- [createData] (object)

- [allowScriptsToClose] (boolean) Allow scripts to close the window.
- [focused] (boolean) If true, opens an active window. If false, opens an inactive window.
- [height] (integer) The height in pixels of the new window, including the frame. If not specified defaults to a natural height.
- [incognito] (boolean) Whether the new window should be an incognito window.
- [left] (integer) The number of pixels to position the new window from the left edge of the screen. If not specified, the new window is offset naturally from the last focused window. This value is ignored for panels.
- [state] (*WindowState*) The initial state of the window. The 'minimized', 'maximized' and 'fullscreen' states cannot be combined with 'left', 'top', 'width' or 'height'.
- [tabId] (integer) The id of the tab for which you want to adopt to the new window.
- [titlePreface] (string) A string to add to the beginning of the window title.
- [top] (integer) The number of pixels to position the new window from the top edge of the screen. If not specified, the new window is offset naturally from the last focused window. This value is ignored for panels.
- [type] (*CreateType*) Specifies what type of browser window to create. The 'panel' and 'detached_panel' types create a popup unless the '–enable-panels' flag is set.
- [url] (string or array of string) A URL or array of URLs to open as tabs in the window. Fully-qualified URLs must include a scheme (i.e. 'http://www.google.com', not 'www.google.com'). Relative URLs will be relative to the current page within the extension. Defaults to the New Tab Page.
- [width] (integer) The width in pixels of the new window, including the frame. If not specified defaults to a natural width.

Returns a Promise fulfilled with:

- *Window* Contains details about the created window.

18.1.6 update(windowId, updateInfo)

Updates the properties of a window. Specify only the properties that you want to change; unspecified properties will be left unchanged.

- windowId (integer)
- updateInfo (object)
  - [drawAttention] (boolean) If true, causes the window to be displayed in a manner that draws the user's attention to the window, without changing the focused window. The effect lasts until the user changes focus to the window. This option has no effect if the window already has focus. Set to false to cancel a previous draw attention request.
  - [focused] (boolean) If true, brings the window to the front. If false, brings the next window in the z-order to the front.
  - [height] (integer) The height to resize the window to in pixels. This value is ignored for panels.
  - [left] (integer) The offset from the left edge of the screen to move the window to in pixels. This value is ignored for panels.
  - [state] (*WindowState*) The new state of the window. The 'minimized', 'maximized' and 'fullscreen' states cannot be combined with 'left', 'top', 'width' or 'height'.
  - [titlePreface] (string) A string to add to the beginning of the window title.

  - [top] (integer) The offset from the top edge of the screen to move the window to in pixels. This value is ignored for panels.
  - [width] (integer) The width to resize the window to in pixels. This value is ignored for panels.

Returns a Promise fulfilled with:

- *Window*

18.1.7 remove(windowId)

Removes (closes) a window, and all the tabs inside it.

- windowId (integer)

18.2 Events

18.2.1 onCreated(window)

Fired when a window is created.

- window (*Window*) Details of the window that was created.

18.2.2 onRemoved(windowId)

Fired when a window is removed (closed).

- windowId (integer) ID of the removed window.

18.2.3 onFocusChanged(windowId)

Fired when the currently focused window changes. Will be *WINDOW_ID_NONE*) if all browser windows have lost focus. Note: On some Linux window managers, WINDOW_ID_NONE will always be sent immediately preceding a switch from one browser window to another.

- windowId (integer) ID of the newly focused window.

18.3 Properties

18.3.1 WINDOW_ID_CURRENT

The windowId value that represents the current window.

18.3.2 WINDOW_ID_NONE

The windowId value that represents the absence of a window.

18.4 Types

18.4.1 CreateType

Specifies what type of browser window to create. The 'panel' and 'detached_panel' types create a popup unless the '–enable-panels' flag is set.

string

Values for CreateType:

- `normal`
- `popup`
- `panel`
- `detached_panel`

18.4.2 Window

object

- `alwaysOnTop` (boolean) Whether the window is set to be always on top.
- `focused` (boolean) Whether the window is currently the focused window.
- `incognito` (boolean) Whether the window is incognito.
- [`height`] (integer) The height of the window, including the frame, in pixels.
- [`id`] (integer) The ID of the window. Window IDs are unique within a session.
- [`left`] (integer) The offset of the window from the left edge of the screen in pixels.
- [`state`] (*WindowState*) The state of this browser window.
- [`tabs`] (array of *Tab*) Array of *Tab* objects representing the current tabs in the window.
- [`title`] (string) The title of the window. Read-only.
- [`top`] (integer) The offset of the window from the top edge of the screen in pixels.
- [`type`] (*WindowType*) The type of browser window this is.
- [`width`] (integer) The width of the window, including the frame, in pixels.

18.4.3 WindowState

The state of this window.

string

Values for WindowState:

- `normal`
- `minimized`
- `maximized`
- `fullscreen`
- `docked`

18.4.4 WindowType

The type of window this is. Under some circumstances a Window may not be assigned type property.

string

Values for WindowType:

- `normal`
- `popup`
- `panel`
- `app`
- `devtools`
- `addressBook` *added in Thunderbird 70, backported to 68.1.1*
- `messageCompose` *added in Thunderbird 70, backported to 68.1.1*
- `messageDisplay` *added in Thunderbird 70, backported to 68.1.1*

The following APIs are also included and work as they do in Firefox:

- contentScripts
- experiments
- extension
- i18n
- management
- permissions
- pkcs11
- runtime
- theme

CHAPTER 19

How To

19.1 WebExtension Experiments

A WebExtension experiment is an additional API that is shipped with your WebExtension. It allows your extension to interact with Thunderbird, much like earlier types of extension did. If you find the built-in Thunderbird APIs can do 80% of what you want to achieve, then WebExtension experiments are for you.

Note: Firefox does not allow WebExtension experiments on release or beta versions. Thunderbird does.

19.1.1 Adding an experiment to your extension

The full code of this example is on GitHub.

Note: This is a very cut-down example. You may find the Firefox documentation helpful, particularly the pages on API schemas, implementing a function, and implementing an event.

Extension manifest

Experimental APIs are declared in the experiment_apis property in a WebExtension's manifest.json file. For example:

```
{
  "manifest_version": 2,
  "name": "Extension containing an experimental API",
  "experiment_apis": {
    "myapi": {
      "schema": "schema.json",
      "parent": {
        "scopes": ["addon_parent"],
```

(continues on next page)

(continued from previous page)

```
        "paths": [["myapi"]],
        "script": "implementation.js"
      }
    }
  }
}
```

Schema

The schema defines the interface between your experiment API and the rest of your extension, which would use `browser.myapi` in this example. In it you describe the functions, events, and types you'll be implementing:

```
[
  {
    "namespace": "myapi",
    "functions": [
      {
        "name": "sayHello",
        "type": "function",
        "description": "Says hello to the user.",
        "async": true,
        "parameters": [
          {
            "name": "name",
            "type": "string",
            "description": "Who to say hello to."
          }
        ]
      }
    ]
  }
]
```

You can see some more-complicated schemas in the Thunderbird source code.

Implementing functions

And finally, the implementation. In this file, you'll put all the code that directly interacts with Thunderbird UI or components. Start by creating an object with the same name as your api at the top level. (Remember to use `var myapi` or `this.myapi`, not `let myapi` or `const myapi`.)

The object has a function `getAPI` which returns another object containing your API. Your functions and events are members of this returned object:

```
var myapi = class extends ExtensionCommon.ExtensionAPI {
  getAPI(context) {
    return {
      myapi: {
        async sayHello(name) {
          Services.wm.getMostRecentWindow("mail:3pane").alert("Hello " + name + "!");
        },
      }
    }
```

(continues on next page)

(continued from previous page)

```
  }
};
```

(Note that the sayHello function is an async function, and `alert` blocks until the prompt is closed. If you call `browser.myapi.sayHello()`, it would return a Promise that doesn't resolve until the user closes the alert.)

Implementing events

The code for events is more complicated, but the pattern is the same every time. The interesting bit is the `register` function, with the argument named `fire` in this example. Any call to `fire.async` will notify listeners that the event fired with the arguments you used.

In `register`, add event listeners, notification observers, or whatever else is needed. `register` runs when the extension calls `browser.myapi.onToolbarClick.addListener`, and returns a function that removes the listeners and observers. This returned function runs when the extension calls `browser.myapi.onToolbarClick.removeListener`, or shuts down.

```
var myapi = class extends ExtensionCommon.ExtensionAPI {
  getAPI(context) {
    return {
      myapi: {
        onToolbarClick: new ExtensionCommon.EventManager({
          context,
          name: "myapi.onToolbarClick",
          register(fire) {
            function callback(event, id, x, y) {
              return fire.async(id, x, y);
            }

            windowListener.add(callback);
            return function() {
              windowListener.remove(callback);
            };
          },
        }).api(),
      }
    }
  }
};
```

Using folder and message types

The built-in schema define some common objects that you may wish to return, namely *MailFolder*, *MessageHeader*, and *MessageList*.

To use these types, interact with the `folderManager` or `messageManager` objects which are members of the `context.extension` object passed to `getAPI`:

```
// Get an nsIMsgFolder from a MailFolder:
let realFolder = context.extension.folderManager.get(accountId, path);

// Get a MailFolder from an nsIMsgFolder:
context.extension.folderManager.convert(realFolder);
```

(continues on next page)

(continued from previous page)

```
// Get an nsIMsgDBHdr from a MessageHeader:
let realMessage = context.extension.messageManager.get(messageId);

// Get a MessageHeader from an nsIMsgDBHdr:
context.extension.messageManager.convert(realMessage);

// Start a MessageList from an array or enumerator of nsIMsgDBHdr:
context.extension.messageManager.startMessageList(realFolder.messages);
```

19.1.2 Experiment API Generator

Try the Experiment Generator to quickly get started making a WebExtension experiment. It doesn't cover all the possibilities, but should be useful for most use-cases or learning how things work.

19.1.3 Getting your API added to Thunderbird

If you think your API would be useful to other extensions, consider having it added to Thunderbird. File a bug that blocks bug 1396172 and add your schema and implementation files as attachments.

19.2 Working with Message Lists

Mail folders could contain a *lot* of messages: thousands, tens of thousands, or even hundreds of thousands.

It would be a very bad idea to deal with that many messages at once, so the WebExtensions APIs split any response that could contain many messages into pages (or chunks, or groups, or whatever...). The default size of each page is 100 messages, although this could change and you **must not** rely on that number.

Each page is an object with two properties: `id`, and `messages`. To get the next page, call *continueList(messageListId)* with the `id` property as an argument:

```
let page = await browser.messages.list(folder);
// Do something with page.messages.

while (page.id) {
  page = await browser.messages.continueList(page.id);
  // Do something with page.messages.
}
```

You could also wrap this code in a generator for ease-of-use:

```
async function* listMessages(folder) {
  let page = await browser.messages.list(folder);
  for (let message of page.messages) {
    yield message;
  }

  while (page.id) {
    page = await browser.messages.continueList(page.id);
    for (let message of page.messages) {
      yield message;
    }
```

(continues on next page)

(continued from previous page)

```
    }
}
```