



Element WebApp

Communications Protocol | Version 0.0.0

Author(s): DATRO Consortium

Aug 13, 2021

CONTENTS

1	Release Notes and Notices	1
1.1	Older Versions	1
1.2	Known and Corrected Issues	1
2	app-load	3
2.1	App load order	3
3	config	9
3.1	Configuration	9
3.2	Identity servers	12
3.3	Desktop app configuration	12
3.4	UI Features	12
4	customisations	14
4.1	Customisations	14
5	e2ee	15
5.1	End to end encryption by default	15
6	Disabling encryption by default	16
6.1	Secure backup	16
7	Requiring secure backup	17
8	Preferring setup methods	18
8.1	Compatibility	18
9	feature-flags	19
9.1	Feature flags	19
10	Interaction with spec process	20
11	Starting work on a feature	21
12	Enabling by default on develop and nightly	22
13	Enabling by default on staging, app, and release	23
14	Feature deployed successfully	24
15	jitsi-dev	25
15.1	Jitsi wrapper developer docs	25

16	Brief introduction to widgets	26
17	Brief introduction to integration managers	27
18	Widgets configured by integration managers	28
19	Jitsi widgets from integration managers	29
20	Jitsi widgets generated by Element itself	30
21	The Jitsi wrapper in Element	31
22	jitsi	32
22.1	Jitsi in Element	32
23	Configuring Element to use your self-hosted Jitsi server	33
24	Element Android	34
25	Element iOS	35
26	kubernetes	36
26.1	Running in Kubernetes	36
27	labs	40
27.1	Labs features	40
28	Submit Abuse Report to Moderators MSC3215 support (feature_report_to_moderators)	41
29	Matrix Spaces MSC1772 support (feature_spaces)	42
30	Render LaTeX maths in messages (feature_latex_maths)	43
31	Message pinning (feature_pinning)	44
32	Custom status (feature_custom_status)	45
33	Custom tags (feature_custom_tags)	46
34	Render simple counters in room header (feature_state_counters)	47
35	Multiple integration managers (feature_many_integration_managers)	48
36	New ways to ignore people (feature_mjolnir)	49
37	Verifications in DMs (feature_dm_verification)	50
38	Bridge info tab (feature_bridge_state)	51
39	Presence indicator in room list (feature_presence_in_room_list)	52
40	Custom themes (feature_custom_themes)	53
41	Message preview tweaks	54
42	Communities v2 prototyping (feature_communities_v2_prototypes) [In Development]	55
43	Dehydrated devices (feature_dehydration)	56

44 Do not disturb (feature_dnd)	57
45 Hidden read receipts (feature_hidden_read_receipts)	58
46 memory-profiles-and-leaks	59
47 Memory leaks	60
48 Memory profiles/snapshots	61
48.1 Taking a memory profile (Firefox)	61
48.2 Taking a memory profile (Chrome/Desktop)	61
49 native-node-modules	63
49.1 Native Node Modules	63
50 pr-previews	64
50.1 Pull Request Previews	64
51 FAQs	65
52 review	66
52.1 Review Guidelines	66
53 Code Review	67
54 Code Quality	69
55 Design and Product Review	70
56 iitemskinning thoughts	71
57 theming	73
57.1 Theming Element	73
57.2 Custom Themes	73
58 translating-dev	75
58.1 How to translate Element (Dev Guide)	75
59 Requirements	76
60 Translating strings vs. marking strings for translation	77
61 Adding new strings	78
62 Editing existing strings	79
63 Adding variables inside a string.	80
64 Things to know/Style Guides	81
65 translating	82
65.1 How to translate Element	82
66 Requirements	83
67 Step 0: Join #element-translations:matrix.org	84
68 Step 1: Preparing your Weblate Profile	85

69	How to check if your language already is being translated	86
70	Step 2a: Helping on existing languages.	87
71	Step 2b: Adding a new language	88
72	Document Publisher(s):	89
72.1	DATRO Consortium	89

RELEASE NOTES AND NOTICES

This section provides information about what is new or changed, including urgent issues, documentation updates, maintenance, and new releases.

- ‘Updates’ are the term used to describe significant changes to our public source code and/or records.

This Release (Version 0.0.0)

```
- **2021-Jul-30** - `Integrated video/voice conferencing added e.g. jitsi.$url1 & ↵  
↵element.$url1`
```

1.1 Older Versions

In the table below the last entry displays a link to an archived copy of the last report. To keep the filename from overflowing in the table below the name displayed may differ from the file name. The date the file was archived will differ from the date of the document label, which is its creation date. If you’re viewing this document on a subdomain of *.datro.world* you may need to right-click and select ‘open link in new tab’. In the interim of a bug fix, you can avoid right-clicking all together, by viewing our document library at its original location gui.8cc.online/static/library

Table 1: Older Versions of this Document

Archive Date	Version	Description	Download Link

1.2 Known and Corrected Issues

Below is a table of pending issues which have been reported to our team. When these issues are remedied, or any significant changed made, a new release will be published.

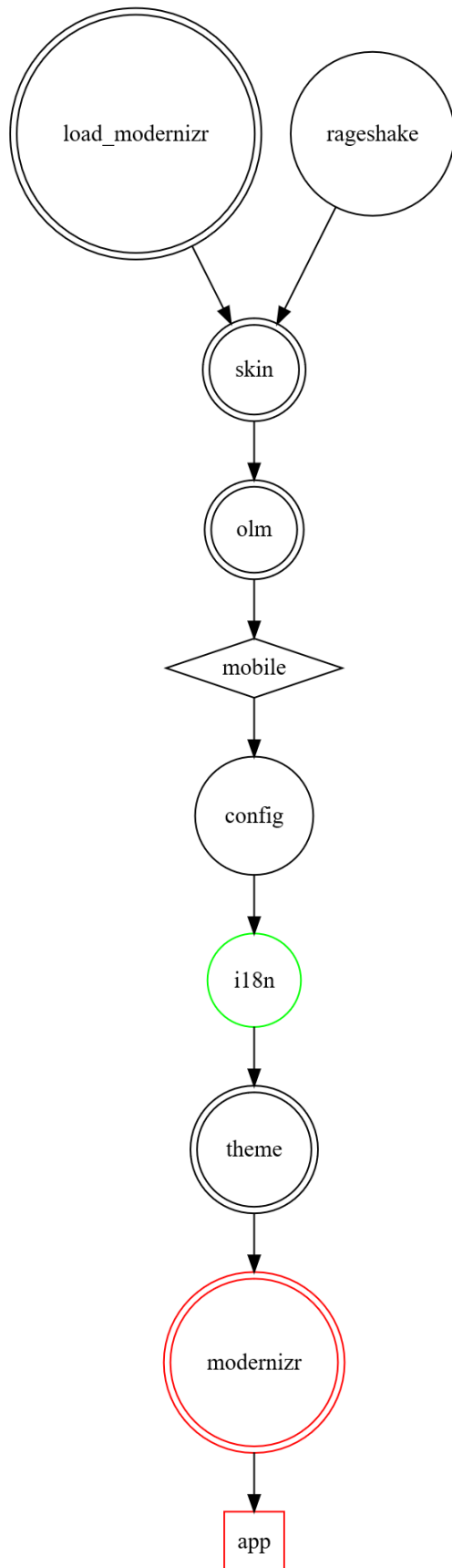
Table 2: Known Issues

file _static/issues.csv			
widths 20	15	25	40
header-rows 1			

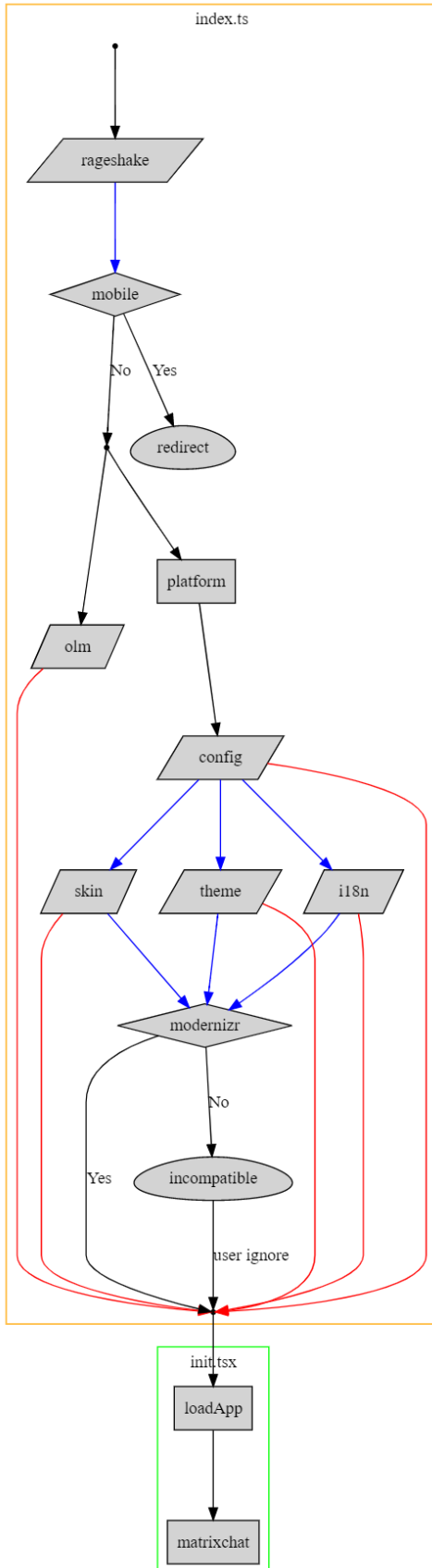
APP-LOAD

2.1 App load order

Old slow flow:



Current more parallel flow:



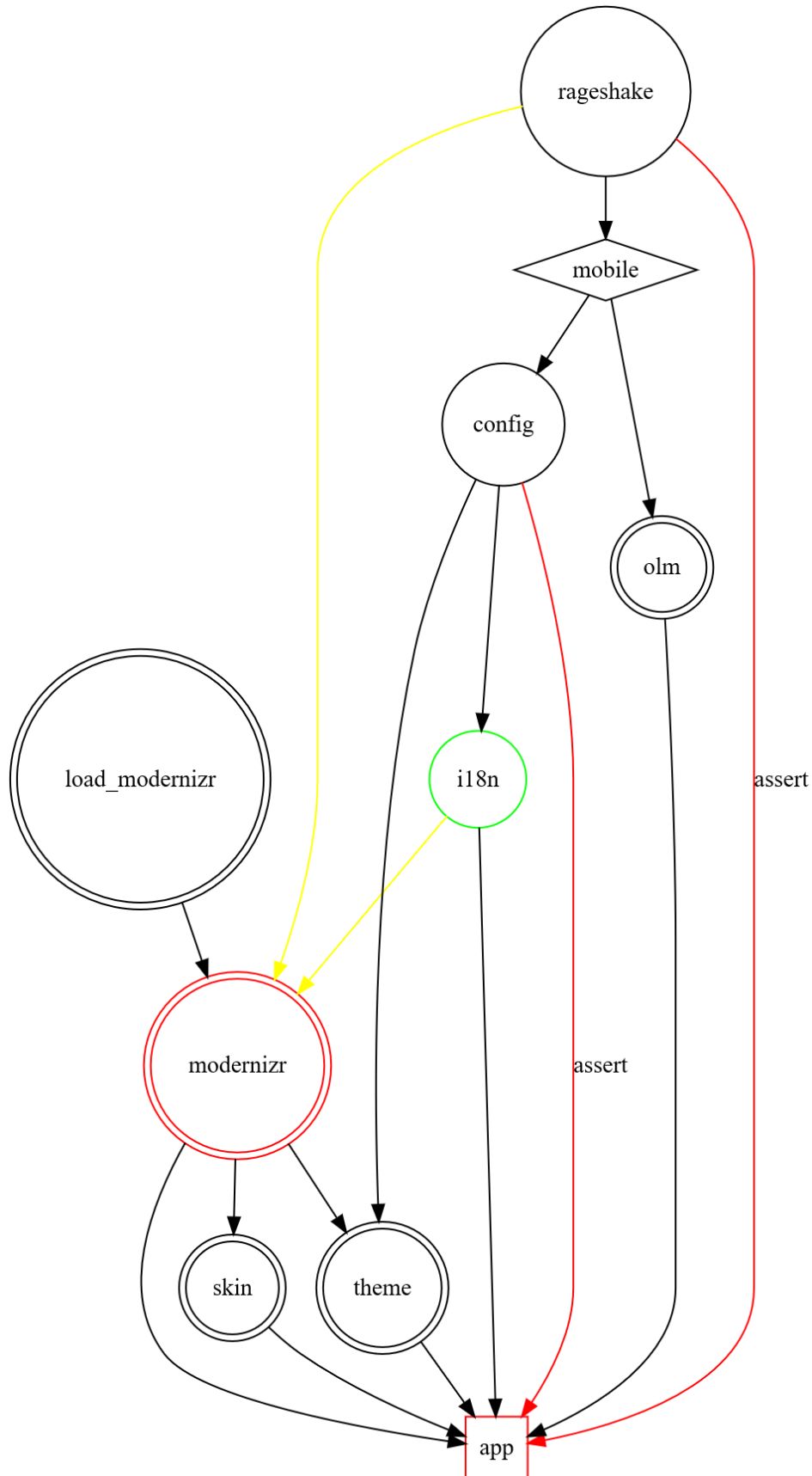
Key:

- Parallelogram: async/await task
- Box: sync task
- Diamond: conditional branch
- Egg: user interaction
- Blue arrow: async task is allowed to settle but allowed to fail
- Red arrow: async task success is asserted

Notes:

- A task begins when all its dependencies (arrows going into it) are fulfilled.
- The success of setting up rageshake is never asserted, element-web has a fallback path for running without IDB (and thus rageshake).
- Everything is awaited to be settled before the Modernizr check, to allow it to make use of things like i18n if they are successful.

Underlying dependencies:



3.1 Configuration

You can configure the app by copying `config.sample.json` to `config.json` and customising it:

For a good example, see <https://develop.element.io/config.json>.

1. `default_server_config` sets the default homeserver and identity server URL for Element to use. The object is the same as returned by `https://raw-html-m2r: '<server_name>/well-known/matrix/client <https://matrix.org/docs/spec/client_server/latest.html#get-well-known-matrix-client>`_`, with added support for a `server_name` under the `m.homeserver` section to display a custom homeserver name. Alternatively, the config can contain a `default_server_name` instead which is where Element will go to get that same object, although this option is deprecated - see the `.well-known` link above for more information on using this option. Note that the `default_server_name` is used to get a complete server configuration whereas the `server_name` in the `default_server_config` is for display purposes only.
 - *Note:* The URLs can also be individually specified as `default_hs_url` and `default_is_url`, however these are deprecated. They are maintained for backwards compatibility with older configurations. `default_is_url` is respected only if `default_hs_url` is used.
 - Element will fail to load if a mix of `default_server_config`, `default_server_name`, or `default_hs_url` is specified. When multiple sources are specified, it is unclear which should take priority and therefore the application cannot continue.
 - As of Element 1.4.0, identity servers are optional. See *Identity servers* below.
2. `sso_immediate_redirect`: When `true`, Element will assume the default server supports SSO and attempt to send the user there to continue (if they aren't already logged in). Default `false`. Note that this disables all usage of the welcome page.
3. `features`: Lookup of optional features that may be force-enabled (`true`) or force-disabled (`false`). When features are not listed here, their defaults will be used, and users can turn them on/off if `showLabsSettings` allows them to. The available optional experimental features vary from release to release and are [documented](#). The feature flag process is [documented](#) as well.
4. `showLabsSettings`: Shows the “labs” tab of user settings. Useful to allow users to turn on experimental features they might not otherwise have access to.
5. `brand`: String to pass to your homeserver when configuring email notifications, to let the homeserver know what email template to use when talking to you.
6. `branding`: Configures various branding and logo details, such as:
 1. `welcomeBackgroundUrl`: An image to use as a wallpaper outside the app during authentication flows. If an array is passed, an image is chosen randomly for each visit.
 2. `authHeaderImageUrl`: An logo image that is shown in the header during authentication flows

3. *authFooterLinks*: a list of links to show in the authentication page footer: [{"text": "Link text", "url": "https://link.target"}, {"text": "Other link", ...}]
7. *reportEvent*: Configures the dialog for reporting content to the homeserver admin.
 1. *adminMessageMD*: An extra message to show on the reporting dialog to mention homeserver-specific policies. Accepts Markdown.
8. *integrations_ui_url*: URL to the web interface for the integrations server. The integrations server is not Element and normally not your homeserver either. The integration server settings may be left blank to disable integrations.
9. *integrations_rest_url*: URL to the REST interface for the integrations server.
10. *integrations_widgets_urls*: list of URLs to the REST interface for the widget integrations server.
11. *bug_report_endpoint_url*: endpoint to send bug reports to (must be running a <https://github.com/matrix-org/rageshake> server). Bug reports are sent when a user clicks “Send Logs” within the application. Bug reports can be disabled/hidden by leaving the *bug_report_endpoint_url* out of your config file.
12. *roomDirectory*: config for the public room directory. This section is optional.
13. *roomDirectory.servers*: List of other homeservers’ directories to include in the drop down list. Optional.
14. *default_theme*: name of theme to use by default (e.g. ‘light’)
15. *update_base_url* (electron app only): HTTPS URL to a web server to download updates from. This should be the path to the directory containing macos and win32 (for update packages, not installer packages).
16. *piwik*: Analytics can be disabled by setting *piwik*: *false* or by leaving the *piwik config option out of your config file*. If you want to enable analytics, set *piwik* to be an object containing the following properties:
 1. *url*: The URL of the Piwik instance to use for collecting analytics
 2. *whitelistedHSUrls*: a list of HS URLs to not redact from the analytics
 3. *whitelistedISUrls*: a list of IS URLs to not redact from the analytics
 4. *siteId*: The Piwik Site ID to use when sending analytics to the Piwik server configured above
17. *welcomeUserId*: the user ID of a bot to invite whenever users register that can give them a tour
18. *embeddedPages*: Configures the pages displayed in portions of Element that embed static files, such as:
 1. *welcomeUrl*: Initial content shown on the outside of the app when not logged in. Defaults to *welcome.html* supplied with Element.
 2. *homeUrl*: Content shown on the inside of the app when a specific room is not selected. By default, no home page is configured. If one is set, a button to access it will be shown in the top left menu.
 3. *loginForWelcome*: Overrides *welcomeUrl* to make the welcome page be the same page as the login page when true. This effectively disables the welcome page.
19. *defaultCountryCode*: The ISO 3166 alpha2 country code to use when showing country selectors, like the phone number input on the registration page. Defaults to GB if the given code is unknown or not provided.
20. *settingDefaults*: Defaults for settings that support the *config* level, as an object mapping setting name to value (note that the “theme” setting is special cased to the *default_theme* in the config file).
21. *disable_custom_urls*: disallow the user to change the default homeserver when signing up or logging in.
22. *permalinkPrefix*: Used to change the URL that Element generates permalinks with. By default, this is “<https://matrix.to>” to generate matrix.to (spec) permalinks. Set this to your Element instance URL if you run an unfederated server (eg: “<https://element.example.org>”).
23. *jitsi*: Used to change the default conference options. Learn more about the Jitsi options at jitsi.md.

1. **preferredDomain**: The domain name of the preferred Jitsi instance. Defaults to `jitsi.riot.im`. This is used whenever a user clicks on the voice/video call buttons - integration managers may use a different domain.
24. **enable_presence_by_hs_url** [The property key should be the URL of the homeserver] and its value defines whether to enable/disable the presence status display from that homeserver. If no options are configured, presence is shown for all homeservers.
25. **disable_guests** [Disables guest access tokens and auto-guest registrations.] Defaults to false (guests are allowed).
26. **disable_login_language_selector** [Disables the login language selector. Defaults] to false (language selector is shown).
27. **disable_3pid_login** [Disables 3rd party identity options on login and registration form] Defaults to false (3rd party identity options are shown).
28. **default_federate** [Default option for room federation when creating a room] Defaults to true (room federation enabled).
29. **desktopBuilds**: Used to alter promotional links to the desktop app. By default the builds are considered available and accessible from <https://element.io>. This config option is typically used in the context of encouraging encrypted message search capabilities (Seshat). All the options listed below are required if this option is specified.
 1. **available**: When false, the desktop app will not be promoted to the user.
 2. **logo**: An HTTP URL to the avatar for the desktop build. Should be 24x24, ideally an SVG.
 3. **url**: An HTTP URL for where to send the user to download the desktop build.
30. **mobileBuilds**: Used to alter promotional links to the mobile app. By default the builds are considered available and accessible from <https://element.io>. This config option is typically used in a context of encouraging the user to try the mobile app instead of a mobile/incompatible browser.
 1. **ios** [The URL to the iOS build. If null, it will be assumed to be not available.] If not set, the default element.io builds will be used.
 2. **android** [The URL to the Android build. If null, it will be assumed to be not available.] If not set, the default element.io builds will be used.
 3. **fdroid**: The URL to the FDroid build. If null, it will be assumed to be not available. If not set, the default element.io builds will be used.
31. **mobileGuideToast**: Whether to show a toast a startup which nudges users on iOS and Android towards the native mobile apps. The toast redirects to the mobile guide if they accept. Defaults to false.
32. **audioStreamUrl**: If supplied, show an option on Jitsi widgets to stream audio using Jitsi's live streaming feature. This option is experimental and may be removed at any time without notice.
33. **voip**: Behaviour related to calls
 1. **obeyAssertedIdentity**: If set, MSC3086 asserted identity messages sent on VoIP calls will cause the call to appear in the room corresponding to the asserted identity. This *must* only be set in trusted environments.
34. **posthog**: [Posthog](<https://posthog.com/>) integration config. If not set, Posthog analytics are disabled.
 1. **projectApiKey**: The Posthog project API key
 2. **apiHost**: The Posthog API host

Note that `index.html` also has an `og:image` meta tag that is set to an image hosted on riot.im. This is the image used if links to your copy of Element appear in some websites like Facebook, and indeed Element itself. This has to be static in the HTML and an absolute URL (and HTTP rather than HTTPS), so it's not possible for this to be an option

in `config.json`. If you'd like to change it, you can build Element, but run `RIOT_OG_IMAGE_URL="http://example.com/logo.png" yarn build`. Alternatively, you can edit the `og:image` meta tag in `index.html` directly each time you download a new version of Element.

3.2 Identity servers

The identity server is used for inviting other users to a room via third party identifiers like emails and phone numbers. It is not used to store your password or account information.

As of Element 1.4.0, all identity server functions are optional and you are prompted to agree to terms before data is sent to the identity server.

Element will check multiple sources when looking for an identity server to use in the following order of preference:

1. The identity server set in the user's account data
 - For a new user, no value is present in their account data. It is only set if the user visits Settings and manually changes their identity server.
2. The identity server provided by the `.well-known` lookup that occurred at login
3. The identity server provided by the Riot config file

If none of these sources have an identity server set, then Element will prompt the user to set an identity server first when attempting to use features that require one.

Currently, the only two public identity servers are <https://vector.im> and <https://matrix.org>, however in the future identity servers will be decentralised.

3.3 Desktop app configuration

See <https://github.com/vector-im/element-desktop#user-specified-configjson>

3.4 UI Features

Parts of the UI can be disabled using UI features. These are settings which appear under `settingDefaults` and can only be `true` (default) or `false`. When `false`, parts of the UI relating to that feature will be disabled regardless of the user's preferences.

Currently, the following UI feature flags are supported:

- `UIFeature.urlPreviews` - Whether URL previews are enabled across the entire application.
- `UIFeature.feedback` - Whether prompts to supply feedback are shown.
- `UIFeature.voip` - Whether or not VoIP is shown readily to the user. When disabled, Jitsi widgets will still work though they cannot easily be added.
- `UIFeature.widgets` - Whether or not widgets will be shown.
- `UIFeature.flair` - Whether or not community flair is shown in rooms.
- `UIFeature.communities` - Whether or not to show any UI related to communities. Implicitly disables `UIFeature.flair` when disabled.
- `UIFeature.advancedSettings` - Whether or not sections titled "advanced" in room and user settings are shown to the user.

- `UIFeature.shareQrCode` - Whether or not the QR code on the share room/event dialog is shown.
- `UIFeature.shareSocial` - Whether or not the social icons on the share room/event dialog are shown.
- `UIFeature.identityServer` - Whether or not functionality requiring an identity server is shown. When disabled, the user will not be able to interact with the identity server (sharing email addresses, 3PID invites, etc).
- `UIFeature.thirdPartyId` - Whether or not UI relating to third party identifiers (3PIDs) is shown. Typically this is considered “contact information” on the homeserver, and is not directly related to the identity server.
- `UIFeature.registration` - Whether or not the registration page is accessible. Typically useful if accounts are managed externally.
- `UIFeature.passwordReset` - Whether or not the password reset page is accessible. Typically useful if accounts are managed externally.
- `UIFeature.deactivate` - Whether or not the deactivate account button is accessible. Typically useful if accounts are managed externally.
- `UIFeature.advancedEncryption` - Whether or not advanced encryption options are shown to the user.
- `UIFeature.roomHistorySettings` - Whether or not the room history settings are shown to the user. This should only be used if the room history visibility options are managed by the server.

CUSTOMISATIONS

4.1 Customisations

Element Web and the React SDK support “customisation points” that can be used to easily add custom logic specific to a particular deployment of Element Web.

An example of this is the [security customisations module](#). This module in the React SDK only defines some empty functions and their types: it does not do anything by default.

To make use of these customisation points, you will first need to fork Element Web so that you can add your own code. Even though the default module is part of the React SDK, you can still override it from the Element Web layer:

1. Copy the default customisation module to `element-web/src/customisations/YourNameSecurity.ts`
2. Edit customisations points and make sure export the ones you actually want to activate
3. Tweak the Element build process to use the customised module instead of the default by adding this to the `additionalPlugins` array in `webpack.config.js`:

```
new webpack.NormalModuleReplacementPlugin(  
  /src[\\\/]customisations[\\\/]Security\.ts/,  
  path.resolve(__dirname, 'src/customisations/YourNameSecurity.ts'),  
)
```

If we add more customisation modules in the future, we’ll likely improve these steps to remove the need for build changes like the above.

By isolating customisations to their own module, this approach should remove the chance of merge conflicts when updating your fork, and thus simplify ongoing maintenance.

5.1 End to end encryption by default

By default, Element will create encrypted DM rooms if the user you are chatting with has keys uploaded on their account. For private room creation, Element will default to encryption on but give you the choice to opt-out.

DISABLING ENCRYPTION BY DEFAULT

Set the following on your homeserver's `/.well-known/matrix/client` config:

```
{
  "io.element.e2ee": {
    "default": false
  }
}
```

6.1 Secure backup

By default, Element strongly encourages (but does not require) users to set up Secure Backup so that cross-signing identity key and message keys can be recovered in case of a disaster where you lose access to all active devices.

REQUIRING SECURE BACKUP

To require Secure Backup to be configured before Element can be used, set the following on your homeserver's `/.well-known/matrix/client` config:

```
{
  "io.element.e2ee": {
    "secure_backup_required": true
  }
}
```

PREFERRING SETUP METHODS

By default, Element offers users a choice of a random key or user-chosen passphrase when setting up Secure Backup. If a homeserver admin would like to only offer one of these, you can signal this via the `/.well-known/matrix/client` config, for example:

```
{
  "io.element.e2ee": {
    "secure_backup_setup_methods": ["passphrase"]
  }
}
```

The field `secure_backup_setup_methods` is an array listing the methods the client should display. Supported values currently include `key` and `passphrase`. If the `secure_backup_setup_methods` field is not present or exists but does not contain any supported methods, Element will fallback to the default value of: `["key", "passphrase"]`.

8.1 Compatibility

The settings above were first proposed under a `im.vector.riot.e2ee` key, which is now deprecated. Element will check for either key, preferring `io.element.e2ee` if both exist.

FEATURE-FLAGS

9.1 Feature flags

When developing new features for Element, we use feature flags to give us more flexibility and control over when and where those features are enabled.

For example, flags make the following things possible:

- Extended testing of a feature via labs on develop
- Enabling features when ready instead of the first moment the code is released
- Testing a feature with a specific set of users (by enabling only on a specific Element instance)

The size of the feature controlled by a feature flag may vary widely: it could be a large project like reactions or a smaller change to an existing algorithm. A large project might use several feature flags if it's useful to control the deployment of different portions independently.

Everyone involved in a feature (engineering, design, product, reviewers) should think about its deployment plan up front as best as possible so we can have the right feature flags in place from the start.

INTERACTION WITH SPEC PROCESS

Historically, we have often used feature flags to guard client features that depend on unstable spec features. Unfortunately, there was never clear agreement about how long such a flag should live for, when it should be removed, etc.

Under the [new spec process](#), server-side unstable features can be used by clients and enabled by default as long as clients commit to doing the associated clean up work once a feature stabilises.

STARTING WORK ON A FEATURE

When starting work on a feature, we should create a matching feature flag:

1. Add a new [setting](#) of the form: .. code-block:: js

```
“feature_cats”: { isFeature: true, displayName: _td(“Adds cats everywhere”), supportedLevels:
    LEVELS_FEATURE, default: false,
},
```

2. Check whether the feature is enabled as appropriate: .. code-block:: js

```
SettingsStore.getValue(“feature_cats”)
```

3. Document the feature in the [labs documentation](#)

With these steps completed, the feature is disabled by default, but can be enabled on develop and nightly by interested users for testing.

Different features may have different deployment plans for when to enable where. The following lists a few common options.

ENABLING BY DEFAULT ON DEVELOP AND NIGHTLY

Set the feature to `true` in the `develop` and `nightly` configs:

```
"features": {  
  "feature_cats": true  
},
```

ENABLING BY DEFAULT ON STAGING, APP, AND RELEASE

Set the feature to `true` in the `staging / app` and `release` configs.

Note: The above will only enable the feature for <https://app.element.io> and official Element Desktop builds. It will not be enabled for self-hosted installed, custom desktop builds, etc. To cover these cases, change the setting's default in `Settings.ts` to `true`.

FEATURE DEPLOYED SUCCESSFULLY

Once we're confident that a feature is working well, we should remove or convert the flag.

If the feature is meant to be turned off/on by the user:

1. Remove `isFeature` from the [setting](#)
2. Change the `default` to `true` (if desired).
3. Remove the feature from the [labs documentation](#)
4. Celebrate!

If the feature is meant to be forced on (non-configurable):

1. Remove the [setting](#)
2. Remove all `getValue` lines that test for the feature.
3. Remove the feature from the [labs documentation](#)
4. If applicable, remove the feature state from [develop](#), [nightly](#), [staging / app](#), and [release](#) configs
5. Celebrate!

15.1 Jitsi wrapper developer docs

If you're looking for information on how to set up Jitsi in your Element, see `jitsi.md` `<./jitsi.md>` instead.

These docs are for developers wondering how the different conference buttons work within Element. If you're not a developer, you're probably looking for [jitsi.md](#).

BRIEF INTRODUCTION TO WIDGETS

Widgets are embedded web applications in a room, controlled through state events, and have a `url` property. They are largely specified by [MSC1236](#) and have extensions proposed under [MSC1286](#).

The `url` is typically something we shove into an `iframe` with sandboxing (see `AppTile` in the `react-sdk`), though for some widgets special integration can be done. v2 widgets have a `data` object which helps achieve that special integration, though v1 widgets are best iframed and left alone.

Widgets have a `postMessage` API they can use to interact with `Element`, which also allows `Element` to interact with them. Typically this is most used by the sticker picker (an account-level widget), though widgets like the Jitsi widget will request permissions to get ‘stuck’ into the room list during a conference.

Widgets can be added with the `/addwidget <url>` command.

BRIEF INTRODUCTION TO INTEGRATION MANAGERS

Integration managers (like Scalar and Dimension) are accessible via the 4 squares in the top right of the room and provide a simple UI over top of bridges, bots, and other stuff to plug into a room. They are a separate service to Element and are thus iframed in a dialog as well. They also have a `postMessage` API they can use to interact with the client to create things like widgets, give permissions to bridges, and generally set everything up for the integration the user is working with.

Integration managers do not currently have a spec associated with them, though efforts are underway in [MSC1286](#).

WIDGETS CONFIGURED BY INTEGRATION MANAGERS

Integration managers will often “wrap” a widget by using a widget url which points to the integration manager instead of to where the user requested the widget be. For example, a custom widget added in an integration manager for <https://matrix.org> will end up creating a widget with a URL like `https://integrations.example.org?widgetUrl=https%3A%2F%2Fmatrix.org`.

The integration manager’s wrapper will typically have another iframe to isolate the widget from the client by yet another layer. The wrapper often provides other functionality which might not be available on the embedded site, such as a fullscreen button or the communication layer with the client (all widgets *should* be talking to the client over `postMessage`, even if they aren’t going to be using the widget APIs).

Widgets added with the `/addwidget` command will *not* be wrapped as they are not going through an integration manager. The widgets themselves *should* also work outside of Element. Widgets currently have a “pop out” button which opens them in a new tab and therefore have no connection back to Riot.

JITSI WIDGETS FROM INTEGRATION MANAGERS

Integration managers will create an entire widget event and send it over `postMessage` for the client to add to the room. This means that the integration manager gets to decide the conference domain, conference name, and other aspects of the widget. As a result, users can end up with a Jitsi widget that does not use the same conference server they specified in their `config.json` - this is expected.

Some integration managers allow the user to change the conference name while others will generate one for the user.

JITSI WIDGETS GENERATED BY ELEMENT ITSELF

When the user clicks on the call buttons by the composer, the integration manager is not involved in the slightest. Instead, Element itself generates a widget event, this time using the `config.json` parameters, and publishes that to the room. If there's only two people in the room, a plain WebRTC call is made instead of using a widget at all - these are defined in the Matrix specification.

The Jitsi widget created by Element uses a local `jitsi.html` wrapper (or one hosted by `https://app.element.io` for desktop users or those on non-https domains) as the widget url. The wrapper has some basic functionality for talking to Element to ensure the required `postMessage` calls are fulfilled.

Note: Per jitsi.md the `preferredDomain` can also come from the server's client `.well-known` data.

THE JITSI WRAPPER IN ELEMENT

Whenever Element sees a Jitsi widget, it ditches the `url` and instead replaces it with its local wrapper, much like what it would do when creating a widget. However, instead of using one from app.element.io, it will use one local to the client instead.

The wrapper is used to provide a consistent experience to users, as well as being faster and less risky to load. The local wrapper URL is populated with the conference information from the original widget (which could be a v1 or v2 widget) so the user joins the right call.

Critically, when the widget URL is reconstructed it does *not* take into account the `config.json`'s `preferredDomain` for Jitsi. If it did this, users would end up on different conference servers and therefore different calls entirely.

Note: Per jitsi.md the `preferredDomain` can also come from the server's client `.well-known` data.

22.1 Jitsi in Element

Element uses [Jitsi](#) for conference calls, which provides options for self-hosting your own server and supports most major platforms.

1:1 calls, or calls between you and one other person, do not use Jitsi. Instead, those calls work directly between clients or via TURN servers configured on the respective homeservers.

There's a number of ways to start a Jitsi call: the easiest way is to click on the voice or video buttons near the message composer in a room with more than 2 people. This will add a Jitsi widget which allows anyone in the room to join.

Integration managers (available through the 4 squares in the top right of the room) may provide their own approaches for adding Jitsi widgets.

CONFIGURING ELEMENT TO USE YOUR SELF-HOSTED JITSİ SERVER

Element will use the Jitsi server that is embedded in the widget, even if it is not the one you configured. This is because conference calls must be held on a single Jitsi server and cannot be split over multiple servers.

However, you can configure Element to *start* a conference with your Jitsi server by adding to your [config](#) the following:

```
{
  "jitsi": {
    "preferredDomain": "your.jitsi.example.org"
  }
}
```

The default is `jitsi.riot.im` (a free service offered by Element), and the demo site for Jitsi uses `meet.jit.si` (also free).

Once you've applied the config change, refresh Element and press the call button. This should start a new conference on your Jitsi server.

Note: The widget URL will point to a `jitsi.html` page hosted by Element. The Jitsi domain will appear later in the URL as a configuration parameter.

Hint: If you want everyone on your homeserver to use the same Jitsi server by default, and you are using `element-web` 1.6 or newer, set the following on your homeserver's `/.well-known/matrix/client` config:

```
{
  "im.vector.riot.jitsi": {
    "preferredDomain": "your.jitsi.example.org"
  }
}
```

ELEMENT ANDROID

Element Android (1.0.5+) supports custom Jitsi domains, similar to Element Web above.

1:1 calls, or calls between you and one other person, do not use Jitsi. Instead, those calls work directly between clients or via TURN servers configured on the respective homeservers.

For rooms with more than 2 joined members, when creating a Jitsi conference via call/video buttons of the toolbar (not via integration manager), Element Android will create a widget using the [wrapper](#) hosted on `app.element.io`. The domain used is the one specified by the `/.well-known/matrix/client` endpoint, and if not present it uses the fallback defined in `config.xml` (`jitsi.riot.im`)

For active Jitsi widgets in the room, a native Jitsi widget UI is created and points to the instance specified in the `domain` key of the widget content data.

Element Android manages allowed native widgets permissions a bit differently than web widgets (as the data shared are different and never shared with the widget URL). For Jitsi widgets, permissions are requested only once per domain (consent saved in account data).

CHAPTER TWENTYFIVE

ELEMENT IOS

Currently the Element mobile apps do not support custom Jitsi servers and will instead use the default `jitsi.riot.im` server. When users on the mobile apps join the call, they will be joining a different conference which has the same name, but not the same participants. This is a known bug and which needs to be fixed.

KUBERNETES

26.1 Running in Kubernetes

In case you would like to deploy element-web in a kubernetes cluster you can use the provided Kubernetes example below as a starting point. Note that this example assumes the Nginx ingress to be installed.

Note that the content of the required `config.json` is defined inside this yaml because it needs to be put in your Kubernetes cluster as a `ConfigMap`.

So to use it you must create a file with this content as a starting point and modify it so it meets the requirements of your environment.

Then you can deploy it to your cluster with something like `kubectl apply -f my-element-web.yaml`.

```
# This is an example of a POSSIBLE config for deploying a single element-web instance in
↳ Kubernetes

# Use the element-web namespace to put it all in.

apiVersion: v1
kind: Namespace
metadata:
  name: element-web

---

# The config.json file is to be put into Kubernetes as a config file in such a way that
# the element web instance can read it.
# The code below shows how this can be done with the config.sample.json content.

apiVersion: v1
kind: ConfigMap
metadata:
  name: element-config
  namespace: element-web
data:
  config.json: |
    {
      "default_server_config": {
        "m.homeserver": {
          "base_url": "https://matrix-client.matrix.org",
          "server_name": "matrix.org"
        }
      }
    }
```

(continues on next page)

(continued from previous page)

```

    },
    "m.identity_server": {
        "base_url": "https://vector.im"
    }
},
"disable_custom_urls": false,
"disable_guests": false,
"disable_login_language_selector": false,
"disable_3pid_login": false,
"brand": "Element",
"integrations_ui_url": "https://scalar.vector.im/",
"integrations_rest_url": "https://scalar.vector.im/api",
"integrations_widgets_urls": [
    "https://scalar.vector.im/_matrix/integrations/v1",
    "https://scalar.vector.im/api",
    "https://scalar-staging.vector.im/_matrix/integrations/v1",
    "https://scalar-staging.vector.im/api",
    "https://scalar-staging.riot.im/scalar/api"
],
"bug_report_endpoint_url": "https://element.io/bugreports/submit",
"defaultCountryCode": "GB",
"showLabsSettings": false,
"features": { },
"default_federate": true,
"default_theme": "light",
"roomDirectory": {
    "servers": [
        "matrix.org"
    ]
},
"piwik": {
    "url": "https://piwik.riot.im/",
    "whitelistedHSUrls": ["https://matrix.org"],
    "whitelistedISUrls": ["https://vector.im", "https://matrix.org"],
    "siteId": 1
},
"enable_presence_by_hs_url": {
    "https://matrix.org": false,
    "https://matrix-client.matrix.org": false
},
"settingDefaults": {
    "breadcrumbs": true
},
"jitsi": {
    "preferredDomain": "jitsi.riot.im"
}
}

```

A deployment of the element-web for a single instance

(continues on next page)

(continued from previous page)

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: element
  namespace: element-web
spec:
  selector:
    matchLabels:
      app: element
  replicas: 1
  template:
    metadata:
      labels:
        app: element
    spec:
      containers:
        - name: element
          image: vectorim/element-web:latest
          volumeMounts:
            - name: config-volume
              mountPath: /app/config.json
              subPath: config.json
          ports:
            - containerPort: 80
              name: element
              protocol: TCP
          readinessProbe:
            httpGet:
              path: /
              port: element
              initialDelaySeconds: 2
              periodSeconds: 3
          livenessProbe:
            httpGet:
              path: /
              port: element
              initialDelaySeconds: 10
              periodSeconds: 10
      volumes:
        - name: config-volume
          configMap:
            name: element-config

```

Wrap it all in a Service

```

apiVersion: v1
kind: Service
metadata:
  name: element

```

(continues on next page)

(continued from previous page)

```
namespace: element-web
spec:
  selector:
    app: element
  ports:
    - name: default
      protocol: TCP
      port: 80
      targetPort: 80

---

# An ingress definition to expose the service via a hostname

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: element
  namespace: element-web
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/configuration-snippet: |
      add_header X-Frame-Options SAMEORIGIN;
      add_header X-Content-Type-Options nosniff;
      add_header X-XSS-Protection "1; mode=block";
      add_header Content-Security-Policy "frame-ancestors 'none'";
spec:
  rules:
    - host: element.example.nl
      http:
        paths:
          - pathType: Prefix
            path: /
            backend:
              service:
                name: element
                port:
                  number: 80

---
```

27.1 Labs features

If Labs is enabled in the [Element config](#), you can enable some of these features by going to Settings->Labs. This list is non-exhaustive and subject to change, chat in [#element-web:matrix.org](#) for more information.

Be warned! Labs features are not finalised, they may be fragile, they may change, they may be dropped. Ask in the room if you are unclear about any details here.

SUBMIT ABUSE REPORT TO MODERATORS MSC3215 SUPPORT (FEATURE_REPORT_TO_MODERATORS)

A new version of the “Report” dialog that lets users send abuse reports directly to room moderators, if the room supports it.

MATRIX SPACES MSC1772 SUPPORT (FEATURE_SPACES)

Enables showing, using, creating, and managing spaces. Create Spaces from the all new Space Panel (to left of Room List).

Incompatible with (will disable) `feature_custom_tags`, `feature_communities_v2_prototypes` and stable Communities/Groups support.

Still in heavy development.

RENDER LATEX MATHS IN MESSAGES (FEATURE_LATEX_MATHS)

Enables rendering of LaTeX maths in messages using [KaTeX](#). LaTeX between single dollar-signs is interpreted as inline maths and double dollar-signs as display maths (i.e. centred on its own line).

MESSAGE PINNING (FEATURE_PINNING)

Allows you to pin messages in the room. To pin a message, use the 3 dots to the right of the message and select “Pin”.

CUSTOM STATUS (FEATURE_CUSTOM_STATUS)

An experimental approach for supporting custom status messages across DMs. To set a status, click on your avatar next to the message composer.

CUSTOM TAGS (FEATURE_CUSTOM_TAGS)

An experimental approach for dealing with custom tags. Custom tags will appear in the bottom portion of the community filter panel.

Setting custom tags is not supported by Element.

RENDER SIMPLE COUNTERS IN ROOM HEADER (FEATURE_STATE_COUNTERS)

Allows rendering of labelled counters above the message list.

Once enabled, send a custom state event to a room to set values:

1. In a room, type `/devtools` to bring up the devtools interface
2. Click “Send Custom Event”
3. Toggle from “Event” to “State Event”
4. Set the event type to: `re.jki.counter` and give it a unique key
5. Specify the content in the following format:

```
{
  "link": "",
  "severity": "normal",
  "title": "my counter",
  "value": 0
}
```

That's it. Now should see your new counter under the header.

MULTIPLE INTEGRATION MANAGERS (FEATURE_MANY_INTEGRATION_MANAGERS)

Exposes a way to access all the integration managers known to Element. This is an implementation of [MSC1957](#).

NEW WAYS TO IGNORE PEOPLE (FEATURE_MJOLNIR)

When enabled, a new settings tab appears for users to be able to manage their ban lists. This is a different kind of ignoring where the ignored user's messages still get rendered, but are hidden by default.

Ban lists are rooms within Matrix, proposed as [MSC2313](#). [Mjolnir](#) is a set of moderation tools which support ban lists.

VERIFICATIONS IN DMS (FEATURE_DM_VERIFICATION)

An implementation of [MSC2241](#). When enabled, verification might not work with devices which don't support MSC2241.

This also includes a new implementation of the user & member info panel, designed to share more code between showing community members & room members. Built on top of this new panel is also a new UX for verification from the member panel.

The setting will be removed in a future release, enabling it non-optionally for all users.

BRIDGE INFO TAB (FEATURE_BRIDGE_STATE)

Adds a “Bridge Info” tab to the Room Settings dialog, if a compatible bridge is present in the room. The Bridge info tab pulls information from the `m.bridge` state event ([MSC2346](#)). Since the feature is based upon a MSC, most bridges are not expected to be compatible, and users should not rely on this tab as the single source of truth just yet.

PRESENCE INDICATOR IN ROOM LIST
(FEATURE_PRESENCE_IN_ROOM_LIST)

This adds a presence indicator in the room list next to DM rooms where the other person is online.

CUSTOM THEMES (FEATURE_CUSTOM_THEMES)

Custom themes are possible through Element's [theme support](#), though normally these themes need to be defined in the config for Element. This labs flag adds an ability for end users to add themes themselves by using a URL to the JSON theme definition.

For some sample themes, check out [aaronraimist/element-themes](#).

MESSAGE PREVIEW TWEAKS

To enable message previews for reactions in all rooms, enable `feature_roomlist_preview_reactions_all`. To enable message previews for reactions in DMs, enable `feature_roomlist_preview_reactions_dms`, ignored when it is enabled for all rooms.

COMMUNITIES V2 PROTOTYPING (FEATURE_COMMUNITIES_V2_PROTOTYPES) [IN DEVELOPMENT]

This is a highly experimental implementation for parts of the communities v2 experience. It does not represent what communities v2 will look/feel like and can/will change without notice. Due to the early stages this feature is in and the requirement for a compatible homeserver, we will not be accepting issues or feedback for this functionality at this time.

DEHYDRATED DEVICES (FEATURE_DEHYDRATION)

Allows users to receive encrypted messages by creating a device that is stored encrypted on the server, as described in [MSC2697](#).

DO NOT DISTURB (FEATURE_DND)

Enables UI for turning on “do not disturb” mode for the current device. When DND mode is engaged, popups and notification noises are suppressed. Not perfect, but can help reduce noise.

HIDDEN READ RECEIPTS (FEATURE_HIDDEN_READ_RECEIPTS)

Enables sending hidden read receipts as per [MSC2285](#)

MEMORY-PROFILES-AND-LEAKS

MEMORY LEAKS

Element usually emits slow behaviour just before it is about to crash. Getting a memory snapshot (below) just before that happens is ideal in figuring out what is going wrong.

Common symptoms are clicking on a room and it feels like the tab froze and scrolling becoming jumpy/staggered.

If you receive a white screen (electron) or the chrome crash page, it is likely run out of memory and it is too late for a memory profile. Please do report when this happens though so we can try and narrow down what might have gone wrong.

MEMORY PROFILES/SNAPSHOTS

When investigating memory leaks/problems it's usually important to compare snapshots from different points in the Element session lifecycle. Most importantly, a snapshot to establish the baseline or “normal” memory usage is useful. Taking a snapshot roughly 30-60 minutes after starting Element is a good time to establish “normal” memory usage for the app - anything after that is at risk of hiding the memory leak and anything newer is still in the warmup stages of the app.

Memory profiles can contain sensitive information. If you are submitting a memory profile to us for debugging purposes, please pick the appropriate Element developer and send them over an encrypted private message. *Do not share your memory profile in public channels or with people you do not trust.*

48.1 Taking a memory profile (Firefox)

1. Press CTRL+SHIFT+I (I as in eye).
2. Click the Memory tab.
3. Press the camera icon in the top left of the pane.
4. Wait a bit (coffee is a good option).
5. When the save button appears on the left side of the panel, click it to save the profile locally.
6. Compress the file (gzip or regular zip) to make the file smaller.
7. Send the compressed file to whoever asked for it (if you trust them).

While the profile is in progress, the tab might be frozen or unresponsive.

48.2 Taking a memory profile (Chrome/Desktop)

1. Press CTRL+SHIFT+I (I as in eye).
2. Click the Memory tab.
3. Select “Heap Snapshot” and the app.element.io VM instance (not the indexeddb one).
4. Click “Take Snapshot”.
5. Wait a bit (coffee is a good option).
6. When the save button appears on the left side of the panel, click it to save the profile locally.
7. Compress the file (gzip or regular zip) to make the file smaller.
8. Send the compressed file to whoever asked for it (if you trust them).

While the profile is in progress, the tab might be frozen or unresponsive.

NATIVE-NODE-MODULES

49.1 Native Node Modules














This documentation moved to the ``*element-desktop*` ` <<https://github.com/vector-im/element-desktop/blob/develop/docs/native-node-modules.md>>`_ repository.

PR-PREVIEWS

50.1 Pull Request Previews

Pull requests to the React SDK layer (and in the future other layers as well) automatically set up a preview site with a full deployment of Element with the changes from the pull request added in so that anyone can easily test and review them. This is especially useful for checking visual and interactive changes.

To access the preview site, scroll down to the bottom of the PR where the various CI results are displayed:

	All checks have passed 3 neutral and 12 successful checks	Hide all checks
	 buildkite/matrix-react-sdk/hammer-and-wrench-build — Passed (4 minutes, 30 seconds)	Details
	 buildkite/matrix-react-sdk/jest-tests — Passed (3 minutes, 47 seconds)	Details
	 buildkite/matrix-react-sdk/pipeline — Passed (9 seconds)	Details
	 buildkite/matrix-react-sdk/stylelint-style-lint — Passed (1 minute, 7 seconds)	Details
	 buildkite/matrix-react-sdk/wrench-element-tests — Passed (4 minutes, 52 seconds)	Details
	 netlify/matrix-react-sdk/deploy-preview — Deploy preview ready!	Details

The checks section could be collapsed at first, so you may need to click “Show all checks” to reveal them. Look for an entry that mentions `deploy-preview`. It may be at the end of the list, so you may need scroll a bit to see it. To access the preview site, click the “Details” link in the deploy preview row.

Important: Please always use test accounts when logging into preview sites, as they may contain unreviewed and potentially dangerous code that could damage your account, exfiltrate encryption keys, etc.

FAQS

Yes, they are created for all PRs from any author.

No, there is no expiry date, so they should remain accessible indefinitely, but of course they obviously aren't meant to live beyond the development workflow, so please don't rely on them for anything important. They may disappear at any time without notice.

52.1 Review Guidelines

The following summarises review guidelines that we follow for pull requests in Element Web and other supporting repos. These are just guidelines (not strict rules) and may be updated over time.

CODE REVIEW

When reviewing code, here are some things we look for and also things we avoid:

- Correctness
- Performance
- Accessibility
- Security
- Quality via automated and manual testing
- Comments and documentation where needed
- Sharing knowledge of different areas among the team
- Ensuring it's something we're comfortable maintaining for the long term
- Progress indicators and local echo where appropriate with network activity
- Style nits that are already handled by the linter
- Dramatically increasing scope
- Use empathetic language
 - See also [Mindful Communication in Code Reviews](#) and [How to Do Code Reviews Like a Human](#)
- Authors should prefer smaller commits for easier reviewing and bisection
- Reviewers should be explicit about required versus optional changes
 - Reviews are conversations and the PR author should feel comfortable discussing and pushing back on changes before making them
- Reviewers are encouraged to ask for tests where they believe it is reasonable
- Core team should lead by example through their tone and language
- Take the time to thank and point out good code changes
- Using softer language like “please” and “what do you think?” goes a long way towards making others feel like colleagues working towards a common goal
- Authors should request review from the element-web team by default (if someone on the team is clearly the expert in an area, a direct review request to them may be more appropriate)
- Reviewers should remove the team review request and request review from themselves when starting a review to avoid double review
- If there are multiple related PRs authors should reference each of the PRs in the others before requesting review. Reviewers might start reviewing from different places and could miss other required PRs.

- Avoid force pushing to a PR after the first round of review
- Use the GitHub default of merge commits when landing (avoid alternate options like squash or rebase)
- PR author merges after review (assuming they have write access)
- Assign issues only when in progress to indicate to others what can be picked up

CODE QUALITY

In the past, we have occasionally written different kinds of tests for Element and the SDKs, but it hasn't been a consistent focus. Going forward, we'd like to change that.

- For new features, code reviewers will expect some form of automated testing to be included by default
- For bug fixes, regression tests are of course great to have, but we don't want to block fixes on this, so we won't require them at this time

The above policy is not a strict rule, but instead it's meant to be a conversation between the author and reviewer. As an author, try to think about writing a test when making your next change. As a reviewer, try to think about how you might test the area of code you are reviewing. If the reviewer agrees it would be quite difficult to test some new feature, then it's okay for them to accept the change without tests for now, but we'd eventually like to be more strict about this further down the road.

If you do spot areas that are quite hard to test today, please let us know in [#element-dev:matrix.org](https://github.com/element-dev/matrix.org). We can work on improving the app architecture and testing helpers so that future tests are easier for everyone to write, but we won't know which parts are difficult unless people shout when stumbling through them.

We recognise that this testing policy will slow things down a bit, but overall it should encourage better long-term health of the app and give everyone more confidence when making changes as coverage increases over time.

For changes guarded by a feature flag, we currently lean towards prioritising our ability to evolve quickly using such flags and thus we will not currently require tests to appear at the same time as the initial landing of features guarded by flags, as long as (for new flagged features going forward) the feature author understands that they are effectively deferring part of their work (adding tests) until later and tests are expected to appear before the feature can be enabled by default.

DESIGN AND PRODUCT REVIEW

We want to ensure that all changes to Element fit with our design and product vision. We often request review from those teams so they can provide their perspective.

In more detail, our usual process for changes that affect the UI or alter user functionality is:

- For changes that will go live when merged, always flag Design and Product teams as appropriate
- For changes guarded by a feature flag, Design and Product review is not required (though may still be useful) since we can continue tweaking

As it can be difficult to review design work from looking at just the changed files in a PR, a [preview site](#) that includes your changes will be added automatically so that anyone who's interested can try them out easily.

Before starting work on a feature, it's best to ensure your plan aligns well with our vision for Element. Please chat with the team in [#element-dev:matrix.org](#) before you start so we can ensure it's something we'd be willing to merge.

IIITEMSKINNING THOUGHTS

== Skinning refactor ==

matrix-react-sdk

- base images
- base CSS
- all the components needed to build a workable app (including the top layer)

element-web: the Element skin

- Element-specific classes (e.g. login header/footer)
- Element-specific themes
 - light
 - dark

i.e. the only things which should go into element-web are bits which apply vector-specific skinning specifically “Stuff that any other brand would not want to use. (e.g. Element logos, links, T&Cs)”

- Questions:
 - Electron app? (should probably be a separate repo in its own right? but might as well go here for now)
 - index.html & index.js? (should be in matrix-react-sdk, given the SDK is useless without them?)

ideally matrix-react-sdk itself should ship with a default skin which actually works built in.

status skin (can go in the same app for now)

- has status theme
 - which inherits from Element light theme
 - how do we share graphics between skins?
 - * shove them into react-sdk, or...
 - * guess we do ../../vector/img
 - * this means keeping the skin name in the images (unless /img is a shortcut to the right skin’s images)

out of scope:

- making the components more independent, so they can be used in isolation.
- that said, the bits which should probably be used by being embedded into a different app:
 - login/reg
 - RoomView + RoomSettings

- MessageComposer
- RoomList
- MemberList
- MemberInfo
- Voip UI
- UserSettings

- sharing different js-sdks between the different isolated modules

other changes:

- how do we handle i18n?
 - each skin should really be its own i18n project. As long as all the commonality stuff is in matrix-react-sdk this shouldn't be too bad.
- ability to associate components with a given skin
 - skins/vector/src <- components
 - skins/vector/css
 - skins/vector/img
 - skins/vector/fonts
- gather together themes (per skin) into a single place too
 - skins/vector/themes/foo/css
 - skins/vector/themes/foo/img
 - skins/vector/themes/foo/fonts
- ideally element-web would contain almost nothing but skins/vector directory.
- ability to entirely replace CSS rather than override it for a given theme
 - e.g. if we replace `Login.js` with `StatusLogin.js`, then we should similarly be able to replace `_Login.scss` with `_StatusLogin.scss`.

random thoughts;

- should we be able to change the entire skin at runtime (more like wordpress) - to the extent of replacing entire components?
 - might pose security issues if a theme can be swapped out to replace MatrixChat or other fundamental functionality at runtime
- if so, perhaps skins & themes should converge...

Immediate plan for Status:

- Implement it as a theme for the Element skin
- Ideally move skins to a sensible level (possibly even including src?)

THEMING

57.1 Theming Element

Themes are a very basic way of providing simple alternative look & feels to the Element app via CSS & custom imagery. They are *NOT* to be confused with ‘skins’, which describe apps which sit on top of matrix-react-sdk - e.g. in theory Element itself is a react-sdk skin. As of Jan 2017, skins are not fully supported; Element is the only available skin.

To define a theme for Element:

1. Pick a name, e.g. teal. at time of writing we have light and dark.
2. Fork `src/skins/vector/css/themes/dark.scss` to be `teal.scss`
3. Fork `src/skins/vector/css/themes/_base.scss` to be `_teal.scss`
4. Override variables in `_teal.scss` as desired. You may wish to delete ones which don't differ from `_base.scss`, to make it clear which are being overridden. If every single colour is being changed (as per `_dark.scss`) then you might as well keep them all.
5. Add the theme to the list of entrypoints in `webpack.config.js`
6. Add the theme to the list of themes in matrix-react-sdk's `UserSettings.js`
7. Sit back and admire your handywork.

In future, the assets for a theme will probably be gathered together into a single directory tree.

57.2 Custom Themes

Themes derived from the built in themes may also be defined in settings.

To avoid name collisions, the internal name of a theme is `custom-${theme.name}`. So if you want to set the custom theme below as the default theme, you would use `default_theme: "custom-Electric Blue"`.

eg. in `config.json`:

```
"settingDefaults": {
  "custom_themes": [
    {
      "name": "Electric Blue",
      "is_dark": false,
      "fonts": {
        "faces": [
          {
```

(continues on next page)

(continued from previous page)

```

        "font-family": "Inter",
        "src": [{"url": "/fonts/Inter.ttf", "format": "ttf"}]
      }
    ],
    "general": "Inter, sans",
    "monospace": "'Courier New'"
  },
  "colors": {
    "accent-color": "#3596fc",
    "primary-color": "#368bd6",
    "warning-color": "#ff4b55",
    "sidebar-color": "#27303a",
    "roomlist-background-color": "#f3f8fd",
    "roomlist-text-color": "#2e2f32",
    "roomlist-text-secondary-color": "#61708b",
    "roomlist-highlights-color": "#ffffff",
    "roomlist-separator-color": "#e3e8f0",
    "timeline-background-color": "#ffffff",
    "timeline-text-color": "#2e2f32",
    "timeline-text-secondary-color": "#61708b",
    "timeline-highlights-color": "#f3f8fd",
    "username-colors": ["#ff0000", ...]
    "avatar-background-colors": ["#cc0000", ...]
  }
}, {
  "name": "Deep Purple",
  "is_dark": true,
  "colors": {
    "accent-color": "#6503b3",
    "primary-color": "#368bd6",
    "warning-color": "#b30356",
    "sidebar-color": "#15171b",
    "roomlist-background-color": "#22262e",
    "roomlist-text-color": "#a1b2d1",
    "roomlist-text-secondary-color": "#edf3ff",
    "roomlist-highlights-color": "#343a46",
    "roomlist-separator-color": "#a1b2d1",
    "timeline-background-color": "#181b21",
    "timeline-text-color": "#edf3ff",
    "timeline-text-secondary-color": "#a1b2d1",
    "timeline-highlights-color": "#22262e"
  }
}
]
}

```

username-colors is expected to contain 8 colors. avatar-background-colors is expected to contain 3 colors. Both values are optional and have fallbacks from the built-in theme.

These are exposed as --username-colors_0,... and --avatar-background-colors_0,... respectively in CSS.

All properties in fonts are optional, and will default to the standard Riot fonts.

TRANSLATING-DEV

58.1 How to translate Element (Dev Guide)

REQUIREMENTS

- A working [Development Setup](#)
 - Including up-to-date versions of matrix-react-sdk and matrix-js-sdk
- Latest LTS version of Node.js installed
- Be able to understand English
- Be able to understand the language you want to translate Element into

TRANSLATING STRINGS VS. MARKING STRINGS FOR TRANSLATION

Translating strings are done with the `_t()` function found in `matrix-react-sdk/lib/languageHandler.js`. It is recommended to call this function wherever you introduce a string constant which should be translated. However, translating can not be performed until after the translation system has been initialized. Thus, sometimes translation must be performed at a different location in the source code than where the string is introduced. This breaks some tooling and makes it difficult to find translatable strings. Therefore, there is the alternative `_td()` function which is used to mark strings for translation, without actually performing the translation (which must still be performed separately, and after the translation system has been initialized).

Basically, whenever a translatable string is introduced, you should call either `_t()` immediately OR `_td()` and later `_t()`.

Example:

```
// Module-level constant
const COLORS = {
  '#f8481c': _td('reddish orange'), // Can't call _t() here yet
  '#fc2647': _td('pinky red') // Use _td() instead so the text is picked up for
  ↪ translation anyway
}

// Function that is called some time after i18n has been loaded
function getColorName(hex) {
  return _t(COLORS[hex]); // Perform actual translation here
}
```

ADDING NEW STRINGS

1. Check if the import `import { _t } from 'matrix-react-sdk/lib/languageHandler';` is present. If not add it to the other import statements. Also import `_td` if needed.
2. Add `_t()\` `` to your string. (Don't forget curly braces when you assign an expression to JSX attributes in the render method). If the string is introduced at a point before the translation system has not yet been initialized, use ```_td()``` instead, and call `_t()` at the appropriate time.
3. Run `yarn i18n` to update `src/i18n/strings/en_EN.json`
4. If you added a string with a plural, you can add other English plural variants to `src/i18n/strings/en_EN.json` (remember to edit the one in the same project as the source file containing your new translation).

EDITING EXISTING STRINGS

1. Edit every occurrence of the string inside `_t()` and `_td()` in the JSX files.
2. Run `yarn i18n` to update `src/i18n/strings/en_EN.json`. (Be sure to run this in the same project as the JSX files you just edited.)
3. Run `yarn prunei18n` to remove the old string from `src/i18n/strings/*.json`.

ADDING VARIABLES INSIDE A STRING.

1. Extend your `_t()` ``call. Instead of `_t(STRING)` use `_t(STRING, {})```
2. Decide how to name it. Please think about if the person who has to translate it can understand what it does. E.g. using the name 'recipient' is bad, because a translator does not know if it is the name of a person, an email address, a user ID, etc. Rather use e.g. `recipientEmailAddress`.
3. Add it to the array in `_t` ``for example `_t(STRING, {variable: this.variable})```
4. Add the variable inside the string. The syntax for variables is `%(variable)s`. Please note the *s* at the end. The name of the variable has to match the previous used name.
 - You can use the special count variable to choose between multiple versions of the same string, in order to get the correct pluralization. E.g. `_t('You have %(count)s new messages', { count: 2 })` ``would show 'You have 2 new messages', while `_t('You have %(count)s new messages', { count: 1 })` would show 'You have one new message' (assuming a singular version of the string has been added to the translation file. See above). Passing `incount` ``is much preferred over having an if-statement choose the correct string to use, because some languages have much more complicated plural rules than english (e.g. they might need a completely different form if there are three things rather than two).
 - If you want to translate text that includes e.g. hyperlinks or other HTML you have to also use tag substitution, e.g. `_t('\ :raw-html-m2r:`<a>Click here!`\ ', {}, { 'a': (sub) => :raw-html-m2r:`<a>{sub}` })`. If you don't do the tag substitution you will end up showing literally “ rather than making a hyperlink.
 - You can also use React components with normal variable substitution if you want to insert HTML markup, e.g. `_t('Your email address is %(emailAddress)s', { emailAddress: :raw-html-m2r:`<i>{userEmailAddress}</i>` })`.

THINGS TO KNOW/STYLE GUIDES

- Do not use `_t()` inside `getDefaultProps`: the translations aren't loaded when `getDefaultProps` is called, leading to missing translations. Use `_td()` to indicate that `_t()` will be called on the string later.
- If using translated strings as constants, translated strings can't be in constants loaded at class-load time since the translations won't be loaded. Mark the strings using `_td()` instead and perform the actual translation later.
- If a string is presented in the UI with punctuation like a full stop, include this in the translation strings, since punctuation varies between languages too.
- Avoid "translation in parts", i.e. concatenating translated strings or using translated strings in variable substitutions. Context is important for translations, and translating partial strings this way is simply not always possible.
- Concatenating strings often also introduces an implicit assumption about word order (e.g. that the subject of the sentence comes first), which is incorrect for many languages.
- Translation 'smell test': If you have a string that does not begin with a capital letter (is not the start of a sentence) or it ends with e.g. `'.'` or a preposition (e.g. `'to'`) you should recheck that you are not trying to translate a partial sentence.
- If you have multiple strings, that are almost identical, except some part (e.g. a word or two) it is still better to translate the full sentence multiple times. It may seem like inefficient repetition, but unlike programming where you try to minimize repetition, translation is much faster if you have many, full, clear, sentences to work with, rather than fewer, but incomplete sentence fragments.

TRANSLATING

65.1 How to translate Element

REQUIREMENTS

- Web Browser
- Be able to understand English
- Be able to understand the language you want to translate Element into

STEP 0: JOIN #ELEMENT-TRANSLATIONS:MATRIX.ORG

1. Come and join <https://matrix.to/#/#element-translations:matrix.org> for general discussion
2. Join <https://matrix.to/#/#element-translators:matrix.org> for language-specific rooms
3. Read scrollback and/or ask if anyone else is working on your language, and co-ordinate if needed. In general little-or-no coordination is needed though :)

STEP 1: PREPARING YOUR WEBLATE PROFILE

1. Head to <https://translate.element.io> and register either via Github or email
2. After registering check if you got an email to verify your account and click the link (if there is none head to step 1.4)
3. Log into weblate
4. Head to <https://translate.element.io/accounts/profile/> and select the languages you know and maybe another language you know too.
5. Head to <https://translate.element.io/accounts/profile/#subscriptions> and select Element Web as Project

HOW TO CHECK IF YOUR LANGUAGE ALREADY IS BEING TRANSLATED

Go to <https://translate.element.io/projects/element-web/> and visit the 2 sub-projects. If your language is listed go to Step 2a and if not go to Step 2b

STEP 2A: HELPING ON EXISTING LANGUAGES.

1. Head to one of the projects listed <https://translate.element.io/projects/element-web/>
2. Click on the `translate` button on the right side of your language
3. Fill in the translations in the writeable field. You will see the original English string and the string of your second language above.

Head to the explanations under Steb 2b

STEP 2B: ADDING A NEW LANGUAGE

1. Go to one of the projects listed <https://translate.element.io/projects/element-web/>
2. Click the `Start new translation` button at the bottom
3. Select a language
4. Start translating like in 2a.3
5. Repeat these steps for the other projects which are listed at the link of step 2b.1

The green button let you save our translations directly. Please only use it if you are 100% sure about that translation. If you do not know a translation please **DO NOT** click that button. Use the arrows above the translations field and click to the right.

The yellow button has to be used if you are unsure about the translation but you have a rough idea. It adds a new suggestion to the string which can then be reviewed by others.

These things are variables that are expanded when displayed by Element. They can be room names, usernames or similar. If you find one, you can move to the right place for your language, but not delete it as the variable will be missing if you do.

A special case is `%(urlStart)s` and `%(urlEnd)s` which are used to mark the beginning of a hyperlink (i.e. `` and ``). You must keep these markers surrounding the equivalent string in your language that needs to be hyperlinked.

You can use inside the translation field “Review needed” checkbox. It will be shown as Strings that need to be reviewed.

The official Weblate doc provides some more in-depth explanation on how to do translations and talks about do and don'ts. You can find it at: <https://docs.weblate.org/en/latest/user/translating.html>

DOCUMENT PUBLISHER(S):

72.1 DATRO Consortium