



Labels and moderation

Moderation in Bluesky consists of multiple, stackable systems, including:

1. Network takedowns which filter the content from the APIs
2. Labels placed on content by moderation services
3. User controls such as mutes and blocks

Developers building client applications should understand how to apply labels (#2) and user controls (#3). For more complete details, see the [Labels Specification](#).

Labels

Labels are published by *moderation services*, which are either hardcoded into the application or chosen by the user. They are attached to records in the responses under the `labels` key.

A label is published with the following information:

```
{
  /** DID of the actor who created this label. */
  src: string
  /** AT URI of the record, repository (account), or other resource that
  this label applies to. */
  uri: string
  /** Optionally, CID specifying the specific version of 'uri' resource
  this label applies to. */
  cid?: string
  /** The short string name of the value or type of this label. */
  val: string
  /** If true, this is a negation label, overwriting a previous label. */
  neg?: boolean
  /** Timestamp when this label was created. */
  cts: string
}
```

Label values

The *value* of a label will determine its behavior. Some example label values are `porn`, `gore`, and `spam`.

Label values are strings. They currently must only be lowercase a-z or a dash character `^[a-z-]+$`. Some of them start with `!`, but that can only be used by global label values.

Label values are interpreted by their definitions. Those definitions include these attributes:

- `blurs` which may be `content` or `media` or `none`
- `severity` which may be `alert` or `inform` or `none`
- `defaultSetting` which may be `hide` or `warn` or `ignore`
- `adultOnly` which is boolean

There are other definition attributes, but they are only used by the global label values.

Global label values

There are a few label values which are defined by the protocol. They are:

- `!hide` which puts a generic warning on content that cannot be clicked through, and filters the content from listings. Not configurable by the user.
- `!warn` which puts a generic warning on content but can be clicked through. Not configurable by the user.
- `!no-unauthenticated` which makes the content inaccessible to logged-out users in applications which respect the label.
- `porn` which puts a warning on images and can only be clicked through if the user is 18+ and has enabled adult content.
- `sexual` which behaves like `porn` but is meant to handle less intense sexual content.
- `graphic-media` which behaves like `porn` but is for violence / gore.
- `nudity` which puts a warning on images but isn't 18+ and defaults to ignore.


There are two reasons global label values exist.


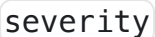
The first is because only label values which are defined globally can be used as self-labels (ie set by a user who is not a Labeler). The `porn`, `sexual`, `gore`, `nudity`, and `!no-unauthenticated` labels are









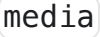







global for this reason.

The second is because some special behaviors, like "non-configurable" and "applies only to logged out users," cannot be applied to custom labels. The !hide, !warn, and !no-unauthenticated labels are global for this reason.

Custom label values

Labelers may define their own label values. Every Labeler has its own namespace of label values it defines. Custom definitions can override all global definitions for the defining Labeler except for the ones that start with a , because those are reserved.

Since there are two behavior attributes ( and ) with three values each, there are 9 possible behaviors for custom label values.

| Blurs | Severity | Description |
|---|---|---|
|  |  | Hide the content and put a "danger" warning label on the content if viewed |
|  |  | Hide the content and put a "neutral" information label on the content if viewed |
|  |  | Hide the content |
|  |  | Hide images in the content and put a "danger" warning label on the content if viewed |
|  |  | Hide images in the content and put a "neutral" information label on the content if viewed |
|  |  | Hide images in the content |
|  |  | Put a "danger" warning label on the content |
|  |  | Put a "neutral" information label on the content |

| Blurs | Severity | Description |
|-------|----------|------------------|
| none | none | No visual effect |

Some examples of the definitions you might use for a label

- Harassment: `blurs=content` + `severity=alert`
- Spider warning: `blurs=media` + `severity=alert`
- Misinformation: `blurs=none` + `severity=alert`
- Verified user: `blurs=none` + `severity=inform`
- Curational down-regulate: `blurs=none` + `severity=none`

The `defaultSetting` establishes how the label will be configured when the user first subscribes to the labeler.

The `adultOnly` establishes whether the label should be configurable if adult content is disabled.

Label targets

Labels may be placed on accounts or records. The Bluesky app interprets the targets in 3 different ways:

- On an account: has account-wide effects
- On a profile: affects only the profile record (e.g. user avatar) and never hides any content, including the account in listings.
- On content: affects the content specifically (a post record, a list, a feedgen, etc)

Labels on accounts or posts have obvious effects. They'll hide the account or post if the user has set the preference to "hide," and they'll blur or warn the content according to the label's settings. A label on an account will cause the warnings to show on all of its posts and other content.

Labels on profiles have much less significant effects. The warnings only show on the profile when viewed directly or in a listing (e.g. "followers"). This means the warnings don't show on the user's posts, unlike when the label is on the account. Most notably: the "hide" preference has no effect when the label is on a profile.

If the label blurs media, a label on a profile will blur the avatar and banner wherever they are shown. This is the main reason you want to apply a label on a profile – because you’re just trying to blur the avatar, and you don’t want to demote the user’s content otherwise.

Label configuration

The user may choose to hide, warn, or ignore each label from a labeler. Hiding and warning are basically similar, except that hide will also filter the labeled content from feeds and listings. Ignore just ignores the label. If adult content is not enabled in preferences, the behavior should force to hide with no override.

Labelers

Labelers publish labels through a labeling service. They also receive reports through a reporting service.

Labelers function like typical atproto accounts. They have a repo, can create posts, and can follow or be followed. An account becomes a Labeler by configuring its DID Document and publishing an `app.bsky.labeler.service` record in their repo.

The labels are not published on the repo. They are signed separately and synced through the labeling service.

Labeler declarations

Labelers publish an `/app.bsky.labeler.service/self` record to declare that they are a labeler and publish their policies. That record looks like this:

```
{
  "$type": "app.bsky.labeler.service",
  "policies": {
    "labelValues": ["porn", "spider"],
    "labelValueDefinitions": [
      {
        "identifier": "spider",
        "severity": "alert",
        "blurs": "media",
        "defaultSetting": "warn",
```

```

    "locales": [
      {
        "lang": "en",
        "name": "Spider Warning",
        "description": "Spider!!!"
      }
    ],
    "subjectTypes": ["record"],
    "subjectCollections": ["app.bsky.feed.post", "app.bsky.actor.profile"],
    "reasonTypes": ["com.atproto.moderation.defs#reason0ther"],
    "createdAt": "2024-03-03T05:31:08.938Z"
  }
}

```

The `labelValues` declares what to expect from the Labeler. It may include global and custom label values.

The `labelValueDefinitions` defines the custom labels. It includes the `locales` field for specifying human-readable copy in various languages. If the user's language is not found, it will use the first set of strings in the array.

`subjectTypes`, `subjectCollections`, and `reasonTypes` declare what type of moderation reports are reviewed by the Labeler. `subjectTypes` can include `record` for individual pieces of content, and `account` for overall accounts. `subjectCollections` is a list of NSIDs of record types; if not defined, any record type is allowed. `reasonTypes` is a list of report reason codes (Lexicon references).

Labeler subscriptions

To include labels from a given labeler, you set the `atproto-accept-labelers` header on the HTTP request to a comma-separated list of the DIDs of the labelers you want to include. The AppView will subscribe to the included labelers (if it hasn't already) and attach any relevant labels. It includes the `atproto-content-labelers` header in the response to indicate which labelers were successfully included.

You may include up to 20 labelers. The Bluesky application hardcodes Bluesky's moderation for inclusion, leaving 19 additional subscriptions to the users.

Self-labels

Users may attach labels to their own content in record types which support it (profile and post are two such examples).

Only global labels can be used for self-labeling, but not all global labels can be used for self-labeling. The supported self-label values are:

- `!no-unauthenticated`
- `porn`
- `sexual`
- `nudity`
- `graphic-media`

While it may be possible to expand the self-labeling vocabulary by adding more global label definitions, we caution against the aggressive policing culture that can form around self-labeling.

Reporting

To send a report to a Labeler, use the `com.atproto.moderation.createReport` procedure. Users may send reports to any of their labelers.

To specify which labeler should receive the label, set the `atproto-proxy` header with the DID of the labeler and the service key of `atproto_labeler`. In the official TypeScript SDK, it looks like this:

```
agent
  .withProxy('atproto_labeler', 'did:web:my-labeler.com')
  .createModerationReport({
    reasonType: 'com.atproto.moderation.defs#reasonViolation',
    reason: 'They were being such a jerk to me!',
    subject: { did: 'did:web:bob.com' },
  })
```

Clients should not send reports to Labelers which do not match the subject and report type metadata in the declaration record (see above).

Official SDK

The official TypeScript SDK provides methods to help developers interact with labels and moderation.

Configuration

Every moderation function takes a set of options which look like this:

```
{
  // the logged-in user's DID
  userDid: 'did:plc:1234...',

  moderationPrefs: {
    // is adult content allowed?
    adultContentEnabled: true,

    // the global label settings (used on self-labels)
    labels: {
      porn: 'hide',
      sexual: 'warn',
      nudity: 'ignore',
      // ...
    },

    // the subscribed labelers and their label settings
    labelers: [
      {
        did: 'did:plc:1234...',
        labels: {
          porn: 'hide',
          sexual: 'warn',
          nudity: 'ignore',
          // ...
        }
      }
    ],

    mutedWords: [/* ... */],
    hiddenPosts: [/* ... */]
  },
}
```



```
// custom label definitions
labelDefs: {
  // labelerDid => defs[]
  'did:plc:1234...': [
    /* ... */
  ]
}
}
```

This should match the following interfaces:

```
export interface ModerationPrefsLabeler {
  did: string
  labels: Record<string, LabelPreference>
}

export interface ModerationPrefs {
  adultContentEnabled: boolean
  labels: Record<string, LabelPreference>
  labelers: ModerationPrefsLabeler[]
  mutedWords: AppBskyActorDefs.MutedWord[]
  hiddenPosts: string[]
}

export interface ModerationOpts {
  userId: string | undefined
  prefs: ModerationPrefs
  /**
   * Map of labeler did -> custom definitions
   */
  labelDefs?: Record<string, InterpretedLabelValueDefinition[]>
}
```

You can quickly grab the `ModerationPrefs` using the `agent.getPreferences()` method:

```
const prefs = await agent.getPreferences()
moderatePost(post, {
  userId: /*...*/,
  prefs: prefs.moderationPrefs,
```

```
  labelDefs: /*...*/  
})
```

To gather the label definitions (`labelDefs`) see the *Labelers* section below.

Labeler management

Labelers are services that provide moderation labels. Your application will typically have 1+ top-level labelers set with the ability to do "takedowns" on content. This is controlled via this static function, though the default is to use Bluesky's moderation:

```
BskyAgent.configure({  
  appLabelers: ['did:web:my-labeler.com'],  
})
```

Users may also add their own labelers. The active labelers are controlled via an HTTP header which is automatically set by the agent when `getPreferences` is called, or when the labeler preferences are changed.

The label value definition are custom labels which only apply to that labeler. Your client needs to sync those definitions in order to correctly interpret them. To do that, call `agent.getLabelers()` or `agent.getLabelDefinitions()` periodically to fetch their definitions. We recommend caching the response (at time our writing the official client uses a TTL of 6 hours).

Here is how to do this:

```
import { BskyAgent } from '@atproto/api'  
  
const agent = new BskyAgent()  
// assume `agent` is a signed in session  
const prefs = await agent.getPreferences()  
const labelDefs = await agent.getLabelDefinitions(prefs)  
  
moderatePost(post, {  
  userId: agent.session.did,  
  prefs: prefs.moderationPrefs,
```

```
labelDefs,
})
```

The `moderate*()` APIs

The SDK exports methods to moderate the different kinds of content on the network.

```
import {
  moderateProfile,
  moderatePost,
  moderateNotification,
  moderateFeedGen,
  moderateUserList,
  moderateLabeler,
} from '@atproto/api'
```

Each of these follows the same API signature:

```
const res = moderatePost(post, moderationOptions)
```

The response object provides an API for figuring out what your UI should do in different contexts.

```
res.ui(context) /* =>
```

```
ModerationUI {
  filter: boolean // should the content be removed from the interface?
  blur: boolean // should the content be put behind a cover?
  alert: boolean // should an alert be put on the content? (negative)
  inform: boolean // should an informational notice be put on the content?
(neutral)
  noOverride: boolean // if blur=true, should the UI disable opening the
cover?

  // the reasons for each of the flags:
  filters: ModerationCause[]
  blurs: ModerationCause[]
  alerts: ModerationCause[]
  informs: ModerationCause[]
```

```
}
*/
```

There are multiple UI contexts available:

- `profileList` A profile being listed, e.g. in search or a follower list
- `profileView` A profile being viewed directly
- `avatar` The user's avatar in any context
- `banner` The user's banner in any context
- `displayName` The user's display name in any context
- `contentList` Content being listed, e.g. posts in a feed, posts as replies, a user list list, a feed generator list, etc
- `contentView` Content being viewed direct, e.g. an opened post, the user list page, the feedgen page, etc
- `contentMedia` Media inside the content, e.g. a picture embedded in a post

Here's how a post in a feed would use these tools to make a decision:

```
const mod = moderatePost(post, moderationOptions)

if (mod.ui('contentList').filter) {
  // dont show the post
}
if (mod.ui('contentList').blur) {
  // cover the post with the explanation from
  mod.ui('contentList').blurs[0]
  if (mod.ui('contentList').noOverride) {
    // dont allow the cover to be removed
  }
}
if (mod.ui('contentMedia').blur) {
  // cover the post's embedded images with the explanation from
  mod.ui('contentMedia').blurs[0]
  if (mod.ui('contentMedia').noOverride) {
    // dont allow the cover to be removed
  }
}
if (mod.ui('avatar').blur) {
  // cover the avatar with the explanation from mod.ui('avatar').blurs[0]
```

```
if (mod.ui('avatar').noOVERRIDE) {  
  // dont allow the cover to be removed  
}  
}  
for (const alert of mod.ui('contentList').alerts) {  
  // render this alert  
}  
for (const inform of mod.ui('contentList').informs) {  
  // render this inform  
}
```

 [Edit this page](#)