Common validation errors when creating issue forms

In this article

You may see some of these common validation errors when creating, saving, or viewing issue forms.

> **Note:** Issue forms are currently in beta and subject to change.

Required top level key `name` is missing

The template does not contain a `name` field, which means it is not clear what to call your issue template when giving users a list of options.

Example of "required top level key `name` is missing" error

```
description: "Thank you for reporting a bug!"
...
```

The error can be fixed by adding `name` as a key.

```
name: "Bug report"
description: "Thank you for reporting a bug!"
...
```

`key` must be a string

This error message means that a permitted key has been provided, but its value cannot be parsed as the data type is not supported.

Example of "`key` must be a string" error

The `description` below is being parsed as a Boolean, but it should be a string.

```
name: "Bug report"
description: true
...
```

The error can be fixed by providing a string as the value. Strings may need to be wrapped in double quotes to be successfully parsed. For example, strings that contain `'` must be wrapped in double quotes.

```
name: "Bug report"
description: "true"
...
```

Empty strings, or strings consisting of only whitespaces, are also not permissible when the field expects a string.

```
name: ""
description: "File a bug report"
assignees: "      "
...
```

The error can be fixed by correcting the value to be a non-empty string. If the field is not required, you should delete the key-value pair.

```
name: "Bug Report"
description: "File a bug report"
...
```

`input` is not a permitted key

An unexpected key was supplied at the top level of the template. For more information about which top-level keys are supported, see "Syntax for issue forms."

Example of "`input` is not a permitted key" error

```
name: "Bug report"
hello: world
...
```

The error can be fixed by removing the unexpected keys.

```
name: "Bug report"
...
```

Forbidden keys

YAML parses certain strings as `Boolean` values. To avoid this, we have explicitly forbidden the usage of the following keys:

`y`, `Y`, `yes`, `Yes`, `YES`, `n`, `N`, `no`, `No`, `NO`, `true`, `True`, `TRUE`, `false`, `False`, `FALSE`, `on`, `On`, `ON`, `off`, `Off`, `OFF`

The error can be fixed by removing the forbidden keys.

Body must contain at least one non-markdown field

Issue forms must accept user input, which means that at least one of its fields must contain a user input field. A `markdown` element is static text, so a `body` array cannot contain only `markdown` elements.

Example of "body must contain at least one non-markdown field" error

```
name: "Bug report"
body:
- type: markdown
  attributes:
    value: "Bugs are the worst!"
```

The error can be fixed by adding non-markdown elements that accept user input.

```
name: "Bug report"
body:
- type: markdown
  attributes:
    value: "Bugs are the worst!"
- type: textarea
  attributes:
    label: "What's wrong?"
```

Body must have unique ids

If using `id` attributes to distinguish multiple elements, each `id` attribute must be unique.

Example of "body must have unique ids" error

```
name: "Bug report"
body:
- type: input
  id: name
  attributes:
    label: First name
- type: input
  id: name
```

```
  attributes:
    label: Last name
```

The error can be fixed by changing the `id` for one of these inputs, so that every `input` field has a unique `id` attribute.

```
name: "Bug report"
body:
- type: input
  id: name
  attributes:
    label: First name
- type: input
  id: surname
  attributes:
    label: Last name
```

Body must have unique labels

When there are multiple `body` elements that accept user input, the `label` attribute for each user input field must be unique.

Example of "body must have unique labels" error

```
name: "Bug report"
body:
- type: textarea
  attributes:
    label: Name
- type: textarea
  attributes:
    label: Name
```

The error can be fixed by changing the `label` attribute for one of the input fields to ensure that each `label` is unique.

```
name: "Bug report"
body:
- type: textarea
  attributes:
    label: Name
- type: textarea
  attributes:
    label: Operating System
```

Input fields can also be differentiated by their `id` attribute. If duplicate `label` attributes are required, you can supply at least one `id` to differentiate two elements with identical labels.

```
name: "Bug report"
body:
- type: textarea
  id: name_1
  attributes:
    label: Name
- type: textarea
  id: name_2
  attributes:
    label: Name
```

`id` attributes are not visible in the issue body. If you want to distinguish the fields in the resulting issue, you should use distinct `label` attributes.

Labels are too similar

Similar labels may be processed into identical references. If an `id` attribute is not provided for an `input`, the `label` attribute is used to generate a reference to the `input` field. To do this, we process the `label` by leveraging the Rails parameterize method. In some cases, two labels that are distinct can be processed into the same parameterized string.

Example of "labels are too similar" error

```
name: "Bug report"
body:
- type: input
  attributes:
    label: Name?
- type: input
  id: name
  attributes:
    label: Name???????
```

The error can be fixed by adding at least one differentiating alphanumeric character, `-`, or `_` to one of the clashing labels.

```
name: "Bug report"
body:
- type: input
  attributes:
    label: Name?
- type: input
  attributes:
    label: Your name
```

The error can also be fixed by giving one of the clashing labels a unique `id`.

```
name: "Bug report"
body:
- type: input
  attributes:
    label: Name?
- type: input
  id: your-name
  attributes:
    label: Name???????
```

Checkboxes must have unique labels

When a `checkboxes` element is present, each of its nested labels must be unique among its peers, as well as among other input types.

Example of "checkboxes must have unique labels" error

```
name: "Bug report"
body:
- type: textarea
  attributes:
```

```
    label: Name
- type: checkboxes
  attributes:
    options:
    - label: Name
```

The error can be fixed by changing the `label` attribute for one of these inputs.

```
name: "Bug report"
body:
- type: textarea
  attributes:
    label: Name
- type: checkboxes
  attributes:
    options:
    - label: Your name
```

Alternatively, you can supply an `id` to any clashing top-level elements. Nested checkbox elements do not support the `id` attribute.

```
name: "Bug report"
body:
- type: textarea
  id: name_1
  attributes:
    label: Name
- type: checkboxes
  attributes:
    options:
    - label: Name
```

`id` attributes are not visible in the issue body. If you want to distinguish the fields in the resulting issue, you should use distinct `label` attributes.

Body[i]: required key type is missing

Each body block must contain the key `type`.

Errors with `body` will be prefixed with `body[i]` where `i` represents the zero-indexed index of the body block containing the error. For example, `body[0]` tells us that the error has been caused by the first block in the `body` list.

Example of "body[i]: required key type is missing" error

```
body:
- attributes:
    value: "Thanks for taking the time to fill out this bug! If you need real-
time help, join us on Discord."
```

The error can be fixed by adding the key `type` with a valid input type as the value. For the available `body` input types and their syntaxes, see "Syntax for GitHub's form schema."

```
body:
- type: markdown
  attributes:
    value: "Thanks for taking the time to fill out this bug! If you need real-
time help, join us on Discord."
```

Body[i]: `x` is not a valid input type

One of the body blocks contains a type value that is not one of the permitted types.

Errors with `body` will be prefixed with `body[i]` where `i` represents the index of the body block containing the error. For example, `body[0]` tells us that the error has been caused by the first block in the `body` list.

Example of "body[i]: `x` is not a valid input type" error

```
body:
- type: x
  attributes:
    value: "Thanks for taking the time to fill out this bug! If you need real-
time help, join us on Discord."
```

The error can be fixed by changing `x` to one of the valid types.

```
body:
- type: markdown
  attributes:
    value: "Thanks for taking the time to fill out this bug! If you need real-
time help, join us on Discord."
```

Body[i]: required attribute key `value` is missing

One of the required `value` attributes has not been provided. The error occurs when a block does not have an `attributes` key or does not have a `value` key under the `attributes` key.

Errors with `body` will be prefixed with `body[i]` where `i` represents the index of the body block containing the error. For example, `body[0]` tells us that the error has been caused by the first block in the `body` list.

Example of "body[i]: required attribute key `value` is missing" error

```
body:
- type: markdown
  attributes:
    value: "Thanks for taking the time to fill out this bug! If you need real-
time help, join us on Discord."
- type: markdown
```

The error in this example can be fixed by adding `value` as a key under `attributes` in the second list element of `body`.

```
body:
- type: markdown
  attributes:
    value: "Thanks for taking the time to fill out this bug! If you need real-
time help, join us on Discord."
- type: markdown
  attributes:
    value: "This is working now!"
```

Body[i]: label must be a string

Within its `attributes` block, a value has the wrong data type.

Errors with `body` will be prefixed with `body[i]` where `i` represents the index of the body block containing the error. For example, `body[0]` tells us that the error has been caused by the first block in the `body` list.

Example of "body[i]: label must be a string" error

The `label` below is being parsed as a Boolean, but it should be a string.

```
body:
- type: markdown
  attributes:
    value: "Thanks for taking the time to fill out this bug! If you need real-
time help, join us on Discord."
- type: textarea
  attributes:
    label: Bug Description
- type: textarea
  attributes:
    label: true
```

The error can be fixed by supplying a string value for `label`. If you want to use a `label` value that may be parsed as a Boolean, integer, or decimal, you should wrap the value in quotes. For example, `"true"` or `"1.3"` instead of `true` or `1.3`.

```
- type: markdown
  attributes:
    value: "Thanks for taking the time to fill out this bug! If you need real-
time help, join us on Discord."
- type: textarea
  attributes:
    label: Bug Description
- type: textarea
  attributes:
    label: Environment Details
```

Empty strings, or strings consisting of only whitespaces, are not permissible when an attribute expects a string. For example, `""` or `" "` are not allowed.

If the attribute is required, the value must be a non-empty string. If the field is not required, you should delete the key-value pair.

```
body:
- type: input
  attributes:
    label: "Name"
```

Body[i]: `id` can only contain numbers, letters, -, _

`id` attributes can only contain alphanumeric characters, `-`, and `_`. Your template may include non-permitted characters, such as whitespace, in an `id`.

Errors with `body` will be prefixed with `body[i]` where `i` represents the index of the body block containing the error. For example, `body[0]` tells us that the error has been caused by the first block in the `body` list.

Example of "body[i]: `id` can only contain numbers, letters, -, _" error

```
name: "Bug report"
body:
- type: input
  id: first name
  attributes:
    label: First name
```

The error can be fixed by ensuring that whitespaces and other non-permitted characters are removed from `id` values.

```
name: "Bug report"
body:
- type: input
  id: first-name
  attributes:
    label: First name
```

Body[i]: `x` is not a permitted key

An unexpected key, `x`, was provided at the same indentation level as `type` and `attributes`.

Errors with `body` will be prefixed with `body[i]` where `i` represents the index of the body block containing the error. For example, `body[0]` tells us that the error has been caused by the first block in the `body` list.

Example of "body[i]: `x` is not a permitted key" error

```
body:
- type: markdown
  x: woof
  attributes:
    value: "Thanks for taking the time to fill out this bug! If you need real-
time help, join us on Discord."
```

The error can be fixed by removing extra keys and only using `type`, `attributes`, and `id`.

```
body:
- type: markdown
  attributes:
    value: "Thanks for taking the time to fill out this bug! If you need real-
time help, join us on Discord."
```

Body[i]: `label` contains forbidden word

To minimize the risk of private information and credentials being posted publicly in GitHub Issues, some words commonly used by attackers are not permitted in the `label` of input or textarea elements.

Errors with `body` will be prefixed with `body[i]` where `i` represents the index of the body block containing the error. For example, `body[0]` tells us that the error has been caused by the first block in the `body` list.

Example of "body[i]: `label` contains forbidden word" error

```
body:
- type: markdown
  attributes:
    value: Hello world!
- type: input
  attributes:
    label: Password
```

The error can be fixed by removing terms like "password" from any `label` fields.

```
body:
- type: markdown
  attributes:
    value: Hello world!
- type: input
  attributes:
    label: Username
```

Body[i]: `x` is not a permitted attribute

An invalid key has been supplied in an `attributes` block.

Errors with `body` will be prefixed with `body[i]` where `i` represents the index of the body block containing the error. For example, `body[0]` tells us that the error has been caused by the first block in the `body` list.

Example of "body[i]: `x` is not a permitted attribute" error

```
body:
- type: markdown
  attributes:
    x: "a random key!"
    value: "Thanks for taking the time to fill out this bug!"
```

The error can be fixed by removing extra keys and only using permitted attributes.

```
body:
- type: markdown
  attributes:
    value: "Thanks for taking the time to fill out this bug!"
```

Body[i]: `options` must be unique

For checkboxes and dropdown input types, the choices defined in the `options` array must be unique.

Errors with `body` will be prefixed with `body[i]` where `i` represents the index of the body block containing the error. For example, `body[0]` tells us that the error has been caused by the first block in the `body` list.

Example of "body[i]: `options` must be unique" error

```
body:
- type: dropdown
```

```
  attributes:
    label: Favorite dessert
    options:
      - ice cream
      - ice cream
      - pie
```

The error can be fixed by ensuring that no duplicate choices exist in the `options` array.

```
body:
- type: dropdown
  attributes:
    label: Favorite dessert
    options:
      - ice cream
      - pie
```

Body[i]: `options` must not include the reserved word, none

"None" is a reserved word in an `options` set because it is used to indicate non-choice when a `dropdown` is not required.

Errors with `body` will be prefixed with `body[i]` where `i` represents the index of the body block containing the error. For example, `body[0]` tells us that the error has been caused by the first block in the `body` list.

Example of "body[i]: `options` must not include the reserved word, none" error

```
body:
- type: dropdown
  attributes:
    label: What types of pie do you like?
    options:
      - Steak & Ale
      - Chicken & Leek
      - None
  validations:
    required: true
```

The error can be fixed by removing "None" as an option. If you want a contributor to be able to indicate that they like none of those types of pies, you can additionally remove the `required` validation.

```
body:
- type: dropdown
  attributes:
    label: What types of pie do you like?
    options:
      - Steak & Ale
      - Chicken & Leek
```

In this example, "None" will be auto-populated as a selectable option.

Body[i]: `options` must not include booleans. Please wrap values such as 'yes', and 'true' in quotes

There are a number of English words that become processed into Boolean values by the YAML parser unless they are wrapped in quotes. For dropdown `options`, all items must be strings rather than Booleans.

Errors with `body` will be prefixed with `body[i]` where `i` represents the index of the body block containing the error. For example, `body[0]` tells us that the error has been caused by the first block in the `body` list.

Example of "body[i]: `options` must not include booleans. Please wrap values such as 'yes', and 'true' in quotes" error

```yaml
body:
- type: dropdown
  attributes:
    label: Do you like pie?
    options:
      - Yes
      - No
      - Maybe
```

The error can be fixed by wrapping each offending option in quotes, to prevent them from being processed as Boolean values.

```yaml
body:
- type: dropdown
  attributes:
    label: Do you like pie?
    options:
      - "Yes"
      - "No"
      - Maybe
```

Body cannot be empty

The template body `key:value` pair can not be empty. For more information about which top-level keys are required, see "Syntax for issue forms."

The error can be fixed by adding the `body:` section.

Example of "body cannot be empty" error

```yaml
name: Support Request
description: Something went wrong and you need help?
---
body:
- type: textarea
  attributes:
    label: "What's wrong?"
```

In this example, the error can be fixed by deleting the `---` (document separator) between the headers and the `body` section.

```yaml
name: Support Request
description: Something went wrong and you need help?

body:
- type: textarea
  attributes:
    label: "What's wrong?"
```

Further reading

- YAML
- Syntax for issue forms