# BMO Documentation

The BMO Team

Sep 14, 2023

# CONTENTS

CHAPTER

ONE

# ABOUT THIS DOCUMENTATION

This is the documentation for version 4.2 of Bugzilla, a bug-tracking system from Mozilla. Bugzilla is an enterprise-class piece of software that tracks millions of bugs and issues for thousands of organizations around the world.

The most current version of this document can always be found on the Bugzilla website.

## 1.1 Evaluating Bugzilla

If you want to try out Bugzilla to see if it meets your needs, you can do so on Mozilla's Bugzilla (BMO) test server, though it comes with various Mozilla-specific customizations. The easiest way to explore the admin tools and more is running a minimum local copy of BMO using Vagrant or Docker. We are not offering any online vanilla test environment at this time.

The Bugzilla FAQ may also be helpful, as it answers a number of questions people sometimes have about whether Bugzilla is for them.

## 1.2 Getting More Help

If this document does not answer your questions, we run a Mozilla forum which can be accessed as a newsgroup, mailing list, or over the web as a Google Group. Please search it first, and then ask your question there.

If you need a guaranteed response, commercial support is available for Bugzilla from a number of people and organizations.

## 1.3 Document Conventions

This document uses the following conventions:

**Warning:** This is a warning—something you should be aware of.

**Note:** This is just a note, for your information.

A filename or a path to a filename is displayed like this: `/path/to/filename.ext`

A command to type in the shell is displayed like this: **`command --arguments`**

A sample of code is illustrated like this:

```
First Line of Code
Second Line of Code
...
```

This documentation is maintained in reStructured Text format using the Sphinx documentation system. It has recently been rewritten, so it undoubtedly has bugs. Please file any you find, in the Bugzilla Documentation component in Mozilla's installation of Bugzilla. If you also want to make a patch, that would be wonderful. Changes are best submitted as diffs, attached to a bug. There is a Style Guide to help you write any new text and markup.

## 1.4 License

Bugzilla is free and open source software, which means (among other things) that you can download it, install it, and run it for any purpose whatsoever without the need for license or payment. Isn't that refreshing?

Bugzilla's code is made available under the Mozilla Public License 2.0 (MPL), specifically the variant which is Incompatible with Secondary Licenses. However, again, if you only want to install and run Bugzilla, you don't need to worry about that; it's only relevant if you redistribute the code or any changes you make.

Bugzilla's documentation is made available under the Creative Commons CC-BY-SA International License 4.0, or any later version.

## 1.5 Credits

The people listed below have made significant contributions to the creation of this documentation:

Andrew Pearson, Ben FrantzDale, Byron Jones, Dave Lawrence, Dave Miller, Dawn Endico, Eric Hanson, Gervase Markham, Jacob Steenhagen, Joe Robins, Kevin Brannen, Martin Wulffeld, Matthew P. Barnson, Ron Teitelbaum, Shane Travis, Spencer Smith, Tara Hernandez, Terry Weissman, Vlad Dascalu, Zach Lipton.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

CHAPTER

TWO

# USER GUIDE

## 2.1 Creating an Account

If you want to use a particular installation of Bugzilla, first you need to create an account. Ask the administrator responsible for your installation for the URL you should use to access it. If you're test-driving Bugzilla, you can use one of the installations on Mozilla's Bugzilla (BMO) test server.

The process of creating an account is similar to many other websites.

1. On the home page, click the *New Account* link in the header. Enter your email address, then click the `Send` button.

   **Note:** If the *New Account* link is not available, this means that the administrator of the installation has disabled self-registration. Speak to the administrator to find out how to get an account.

2. Within moments, you should receive an email to the address you provided, which contains your login name (generally the same as the email address), and a URL to click to confirm your registration.
3. Once you confirm your registration, Bugzilla will ask you your real name (optional, but recommended) and ask you to choose a password. Depending on how your Bugzilla is configured, there may be minimum complexity requirements for the password.
4. Now all you need to do is to click the *Log In* link in the header or footer, enter your email address and the password you just chose into the login form, and click the *Log in* button.

You are now logged in. Bugzilla uses cookies to remember you are logged in, so, unless you have cookies disabled or your IP address changes, you should not have to log in again during your session.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 2.2 Filing a Bug

### 2.2.1 Reporting a New Bug

Years of bug writing experience has been distilled for your reading pleasure into the Bug report writing guidelines. While some of the advice is Mozilla-specific, the basic principles of reporting Reproducible, Specific bugs and isolating the Product you are using, the Version of the Product, the Component which failed, the Hardware Platform, and Operating System you were using at the time of the failure go a long way toward ensuring accurate, responsible fixes for the bug that bit you.

**Note:** If you want to file a test bug to see how Bugzilla works, you can do so on Mozilla's Bugzilla (BMO) test server. Please don't do it on any production Bugzilla installation.

The procedure for filing a bug is as follows:

1. Click the *New* link available in the header or footer of pages, or the *File a Bug* link on the home page.
2. First, you have to select the product in which you found a bug.
3. You now see a form where you can specify the component (part of the product which is affected by the bug you discovered; if you have no idea, just select *General* if such a component exists), the version of the program you were using, the operating system and platform your program is running on and the severity of the bug (if the bug you found crashes the program, it's probably a major or a critical bug; if it's a typo somewhere, that's something pretty minor; if it's something you would like to see implemented, then that's an enhancement).
4. You also need to provide a short but descriptive summary of the bug you found. "My program is crashing all the time" is a very poor summary and doesn't help developers at all. Try something more meaningful or your bug will probably be ignored due to a lack of precision. In the Description, give a detailed list of steps to reproduce the problem you encountered. Try to limit these steps to a minimum set required to reproduce the problem. This will make the life of developers easier, and the probability that they consider your bug in a reasonable timeframe will be much higher.

   **Note:** Try to make sure that everything in the Summary is also in the Description. Summaries are often updated and this will ensure your original information is easily accessible.

5. As you file the bug, you can also attach a document (testcase, patch, or screenshot of the problem).
6. Depending on the Bugzilla installation you are using and the product in which you are filing the bug, you can also request developers to consider your bug in different ways (such as requesting review for the patch you just attached, requesting your bug to block the next release of the product, and many other product-specific requests).
7. Now is a good time to read your bug report again. Remove all misspellings; otherwise, your bug may not be found by developers running queries for some specific words, and so your bug would not get any attention. Also make sure you didn't forget any important information developers should know in order to reproduce the problem, and make sure your description of the problem is explicit and clear enough. When you think your bug report is ready to go, the last step is to click the *Submit Bug* button to add your report into the database.

### 2.2.2 Clone an Existing Bug

Bugzilla allows you to "clone" an existing bug. The newly created bug will inherit most settings from the old bug. This allows you to track similar concerns that require different handling in a new bug. To use this, go to the bug that you want to clone, then click the *Clone This Bug* link on the bug page. This will take you to the *Enter Bug* page that is filled with the values that the old bug has. You can then change the values and/or text if needed.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 2.3 Understanding a Bug

The core of Bugzilla is the screen which displays a particular bug. Note that the labels for most fields are hyperlinks; clicking them will take you to context-sensitive help on that particular field. Fields marked * may not be present on every installation of Bugzilla.

***Summary:***
A one-sentence summary of the problem, displayed in the header next to the bug number.

***Status (and Resolution):***
These define exactly what state the bug is in—from not even being confirmed as a bug, through to being fixed and the fix confirmed by Quality Assurance. The different possible values for Status and Resolution on your installation should be documented in the context-sensitive help for those items.

***Alias:***
A unique short text name for the bug, which can be used instead of the bug number.

***Product and Component*:**
Bugs are divided up by Product and Component, with a Product having one or more Components in it.

***Version:***
The "Version" field usually contains the numbers or names of released versions of the product. It is used to indicate the version(s) affected by the bug report.

***Hardware (Platform and OS):***
These indicate the computing environment where the bug was found.

***Importance (Priority and Severity):***
The Priority field is used to prioritize bugs, either by the assignee, or someone else with authority to direct their time such as a project manager. It's a good idea not to change this on other people's bugs. The default values are P1 to P5.

The Severity field indicates how severe the problem is—from blocker ("application unusable") to trivial ("minor cosmetic issue"). You can also use this field to indicate whether a bug is an enhancement request.

****Target Milestone:***
A future version by which the bug is to be fixed. e.g. The Bugzilla Project's milestones for future Bugzilla versions are 4.4, 5.0, 6.0, etc. Milestones are not restricted to numbers, though—you can use any text strings, such as dates.

***Assigned To:***
The person responsible for fixing the bug.

****QA Contact:***
The person responsible for quality assurance on this bug.

***URL:***
A URL associated with the bug, if any.

****Whiteboard:***
A free-form text area for adding short notes and tags to a bug.

***Keywords:***
The administrator can define keywords which you can use to tag and categorize bugs—e.g. `crash` or `regression`.

***Personal Tags:***
Unlike Keywords which are global and visible by all users, Personal Tags are personal and can only be viewed and edited by their author. Editing them won't send any notifications to other users. Use them to tag and keep track of sets of bugs that you personally care about, using your own classification system.

***Dependencies (Depends On and Blocks):***
If this bug cannot be fixed unless other bugs are fixed (depends on), or this bug stops other bugs being fixed (blocks), their numbers are recorded here.

Clicking the *Dependency tree* link shows the dependency relationships of the bug as a tree structure. You can change how much depth to show, and you can hide resolved bugs from this page. You can also collapse/expand dependencies for each non-terminal bug on the tree view, using the [-]/[+] buttons that appear before the summary.

***Opened:***
The person who filed the bug, and the date and time they did it.

***Updated:***
The date and time the bug was last changed.

***CC List:***
A list of people who get mail when the bug changes, in addition to the Reporter, Assignee and QA Contact (if enabled).

***Ignore Bug Mail:***
Set this if you want never to get bugmail from this bug again. See also *Email Preferences*.

****See Also:***
Bugs, in this Bugzilla, other Bugzillas, or other bug trackers, that are related to this one.

***Flags:***
A flag is a kind of status that can be set on bugs or attachments to indicate that the bugs/attachments are in a certain state. Each installation can define its own set of flags that can be set on bugs or attachments. See *Flags*.

****Time Tracking:***
This form can be used for time tracking. To use this feature, you have to be a member of the group specified by the timetrackinggroup parameter. See *Time Tracking* for more information.

**Orig. Est.:**
This field shows the original estimated time.

**Current Est.:**
This field shows the current estimated time. This number is calculated from `Hours Worked` and `Hours Left`.

**Hours Worked:**
This field shows the number of hours worked.

**Hours Left:**
This field shows the `Current Est.` - `Hours Worked`. This value + `Hours Worked` will become the new Current Est.

**%Complete:**
This field shows what percentage of the task is complete.

**Gain:**
This field shows the number of hours that the bug is ahead of the `Orig. Est.`.

**Deadline:**
This field shows the deadline for this bug.

***Attachments:***
You can attach files (e.g. test cases or patches) to bugs. If there are any attachments, they are listed in this section. See *Attachments* for more information.

***Additional Comments:***
You can add your two cents to the bug discussion here, if you have something worthwhile to say.

### 2.3.1 Flags

Flags are a way to attach a specific status to a bug or attachment, either `+` or `-`. The meaning of these symbols depends on the name of the flag itself, but contextually they could mean pass/fail, accept/reject, approved/denied, or even a simple yes/no. If your site allows requestable flags, then users may set a flag to `?` as a request to another user that they look at the bug/attachment and set the flag to its correct status.

A set flag appears in bug reports and on "edit attachment" pages with the abbreviated username of the user who set the flag prepended to the flag name. For example, if Jack sets a "review" flag to +, it appears as *Jack: review [ + ]*.

A requested flag appears with the user who requested the flag prepended to the flag name and the user who has been requested to set the flag appended to the flag name within parentheses. For example, if Jack asks Jill for review, it appears as *Jack: review [ ? ] (Jill)*.

You can browse through open requests made of you and by you by selecting *My Requests* from the footer. You can also look at open requests limited by other requesters, requestees, products, components, and flag names. Note that you can use '-' for requestee to specify flags with no requestee set.

#### A Simple Example

A developer might want to ask their manager, "Should we fix this bug before we release version 2.0?" They might want to do this for a *lot* of bugs, so they decide to streamline the process. So:

1. The Bugzilla administrator creates a flag type called blocking2.0 for bugs in your product. It shows up on the *Show Bug* screen as the text *blocking2.0* with a drop-down box next to it. The drop-down box contains four values: an empty space, `?`, `-`, and `+`.
2. The developer sets the flag to `?`.
3. The manager sees the *blocking2.0* flag with a `?` value.
4. If the manager thinks the feature should go into the product before version 2.0 can be released, they set the flag to `+`. Otherwise, they set it to `-`.
5. Now, every Bugzilla user who looks at the bug knows whether or not the bug needs to be fixed before release of version 2.0.

#### About Flags

Flags can have four values:

**?**
: A user is requesting that a status be set. (Think of it as 'A question is being asked'.)

**-**
: The status has been set negatively. (The question has been answered `no`.)

**+**
: The status has been set positively. (The question has been answered `yes`.)

**_**
: `unset` actually shows up as a blank space. This just means that nobody has expressed an opinion (or asked someone else to express an opinion) about the matter covered by this flag.

### Flag Requests

If a flag has been defined as *requestable*, and a user has enough privileges to request it (see below), the user can set the flag's status to `?`. This status indicates that someone (a.k.a. "the requester") is asking someone else to set the flag to either + or `-`.

If a flag has been defined as *specifically requestable*, a text box will appear next to the flag into which the requester may enter a Bugzilla username. That named person (a.k.a. "the requestee") will receive an email notifying them of the request, and pointing them to the bug/attachment in question.

If a flag has *not* been defined as *specifically requestable*, then no such text box will appear. A request to set this flag cannot be made of any specific individual; these requests are open for anyone to answer. In Bugzilla this is known as "asking the wind". A requester may ask the wind on any flag simply by leaving the text box blank.

### Attachment Flags

There are two types of flags: bug flags and attachment flags.

Attachment flags are used to ask a question about a specific attachment on a bug.

Many Bugzilla installations use this to request that one developer review another developer's code before they check it in. They attach the code to a bug report, and then set a flag on that attachment called *review* to *review? reviewer@example.com*. reviewer@example.com is then notified by email that they have to check out that attachment and approve it or deny it.

For a Bugzilla user, attachment flags show up in three places:

1. On the list of attachments in the *Show Bug* screen, you can see the current state of any flags that have been set to ?, +, or `-`. You can see who asked about the flag (the requester), and who is being asked (the requestee).
2. When you edit an attachment, you can see any settable flag, along with any flags that have already been set. The *Edit Attachment* screen is where you set flags to ?, `-`, +, or unset them.
3. Requests are listed in the *Request Queue*, which is accessible from the *My Requests* link (if you are logged in) or *Requests* link (if you are logged out) visible on all pages.

### Bug Flags

Bug flags are used to set a status on the bug itself. You can see Bug Flags in the *Show Bug* and *Requests* screens, as described above.

Only users with enough privileges (see below) may set flags on bugs. This doesn't necessarily include the assignee, reporter, or users with the editbugs permission.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 2.4 Editing a Bug

### 2.4.1 Attachments

Attachments are used to attach relevant files to bugs - patches, screenshots, test cases, debugging aids or logs, or anything else binary or too large to fit into a comment.

You should use attachments, rather than comments, for large chunks of plain text data, such as trace, debugging output files, or log files. That way, it doesn't bloat the bug for everyone who wants to read it, and cause people to receive large, useless mails.

You should make sure to trim screenshots. There's no need to show the whole screen if you are pointing out a single-pixel problem.

Bugzilla stores and uses a Content-Type for each attachment (e.g. text/html). To download an attachment as a different Content-Type (e.g. application/xhtml+xml), you can override this using a 'content_type' parameter on the URL, e.g. `&content_type=text/plain`.

Also, you can enter the URL pointing to the attachment instead of uploading the attachment itself. For example, this is useful if you want to point to an external application, a website or a very large file.

It's also possible to create an attachment by pasting text directly in a text field; Bugzilla will convert it into an attachment. This is pretty useful when you are copying and pasting, to avoid the extra step of saving the text in a temporary file.

### 2.4.2 Flags

To set a flag, select either + or - from the drop-down menu next to the name of the flag in the *Flags* list. The meaning of these values are flag-specific and thus cannot be described in this documentation, but by way of example, setting a flag named *review* + may indicate that the bug/attachment has passed review, while setting it to - may indicate that the bug/attachment has failed review.

To unset a flag, click its drop-down menu and select the blank value. Note that marking an attachment as obsolete automatically cancels all pending requests for the attachment.

If your administrator has enabled requests for a flag, request a flag by selecting *?* from the drop-down menu and then entering the username of the user you want to set the flag in the text field next to the menu.

### 2.4.3 Time Tracking

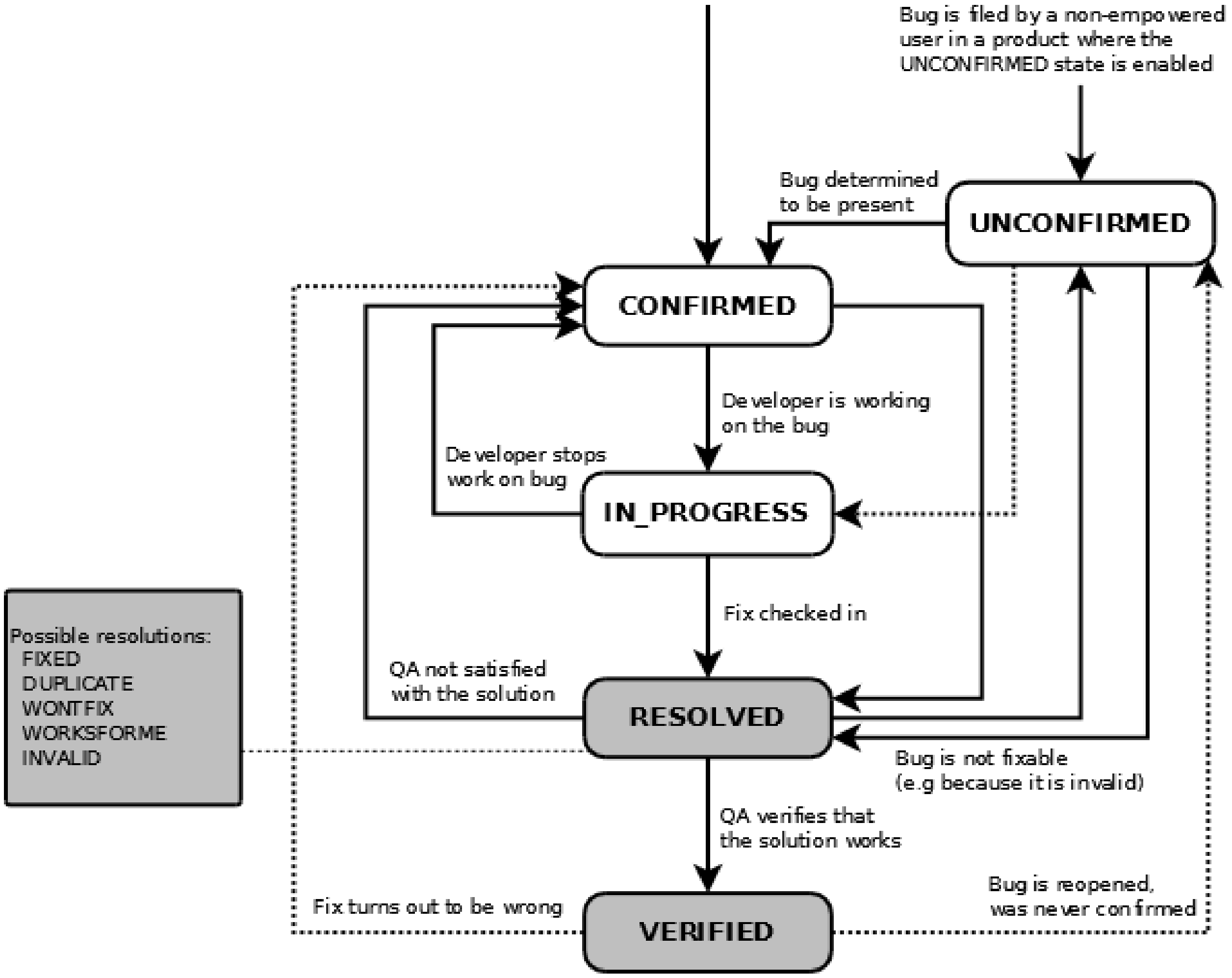
Users who belong to the group specified by the `timetrackinggroup` parameter have access to time-related fields. Developers can see deadlines and estimated times to fix bugs, and can provide time spent on these bugs. Users who do not belong to this group can only see the deadline but not edit it. Other time-related fields remain invisible to them.

At any time, a summary of the time spent by developers on bugs is accessible either from bug lists when clicking the `Time Summary` button or from individual bugs when clicking the `Summarize time` link in the time tracking table. The `summarize_time.cgi` page lets you view this information either per developer or per bug and can be split on a month basis to have greater details on how time is spent by developers.

As soon as a bug is marked as RESOLVED, the remaining time expected to fix the bug is set to zero. This lets QA people set it again for their own usage, and it will be set to zero again when the bug is marked as VERIFIED.

### 2.4.4 Life Cycle of a Bug

The life cycle of a bug, also known as workflow, is customizable to match the needs of your organization (see *Workflow*). The image below contains a graphical representation of the default workflow using the default bug statuses. If you wish to customize this image for your site, the diagram file is available in Dia's native XML format.


Bug is filed by a non-empowered user in a product where the UNCONFIRMED state is enabled
UNCONFIRMED
Bug determined to be present
CONFIRMED
Developer is working on the bug
Developer stops work on bug
IN_PROGRESS
Fix checked in
QA not satisfied with the solution
RESOLVED
Bug is not fixable (e.g because it is invalid)
QA verifies that the solution works
Bug is reopened, was never confirmed
Fix turns out to be wrong
VERIFIED
Possible resolutions:
FIXED
DUPLICATE
WONTFIX
WORKSFORME
INVALID

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 2.5 Finding Bugs

Bugzilla has a number of different search options.

**Note:** Bugzilla queries are case-insensitive and accent-insensitive when used with either MySQL or Oracle databases. When using Bugzilla with PostgreSQL, however, some queries are case sensitive. This is due to the way PostgreSQL handles case and accent sensitivity.

### 2.5.1 Quicksearch

Quicksearch is a single-text-box query tool. You'll find it in Bugzilla's header or footer.

Quicksearch uses metacharacters to indicate what is to be searched. For example, typing

```
foo|bar
```

into Quicksearch would search for "foo" or "bar" in the summary and status whiteboard of a bug; adding

```
:BazProduct
```

would search only in that product.

You can also use it to go directly to a bug by entering its number or its alias.

### 2.5.2 Simple Search

Simple Search is good for finding one particular bug. It works like internet search engines - just enter some keywords and off you go.

### 2.5.3 Advanced Search

The Advanced Search page is used to produce a list of all bugs fitting exact criteria. You can play with it on Mozilla's Bugzilla (BMO) test server.

Advanced Search has controls for selecting different possible values for all of the fields in a bug, as described above. For some fields, multiple values can be selected. In those cases, Bugzilla returns bugs where the content of the field matches any one of the selected values. If none is selected, then the field can take any value.

After a search is run, you can save it as a Saved Search, which will appear in the page footer. If you are in the group defined by the "querysharegroup" parameter, you may share your queries with other users; see *Saved Searches* for more details.

### 2.5.4 Custom Search

Highly advanced querying is done using the *Custom Search* feature of the *Advanced Search* page.

The search criteria here further restrict the set of results returned by a query, over and above those defined in the fields at the top of the page. It is thereby possible to search for bugs based on elaborate combinations of criteria.

The simplest custom searches have only one term. These searches permit the selected *field* to be compared using a selectable *operator* to a specified *value*. Much of this could be reproduced using the standard fields. However, you can then combine terms using "Match All" (AND) or "Match Any" (OR), using groups for combining and priority, in order to construct searches of almost arbitrary complexity.

There are three fields in each row (known as a "term") of a custom search:

- *Field:* the name of the field being searched
- *Operator:* the comparison operator
- *Value:* the value to which the field is being compared

The list of available *fields* contains all the fields defined for a bug, including any custom fields, and then also some pseudo-fields like *Assignee Real Name*, *Days Since Bug Changed*, *Time Since Assignee Touched* and other things it may be useful to search on.

There are a wide range of *operators* available, not all of which may make sense for a particular field. There are various string-matching operations (including regular expressions), numerical comparisons (which also work for dates), and also the ability to search for change information—when a field changed, what it changed from or to, and who did it. There are special operators for *is empty* and *is not empty*, because Bugzilla can't tell the difference between a value field left blank on purpose and one left blank by accident.

You can have an arbitrary number of rows and groups, and rearrange them by dragging and dropping the handle on each item. You can even duplicate an item by holding the Alt key while dragging it. The radio buttons above them define how they relate — *Match All*, *Match All (Same Field)* or *Match Any*. The difference between the first and second can be illustrated with a comment search. If you have a search:

```
Comment   contains the string   "Fred"
Comment   contains the string   "Barney"
```

then under the first regime (match separately) the search would return bugs where "Fred" appeared in one comment and "Barney" in the same or any other comment, whereas under the second (match against the same field), both strings would need to occur in exactly the same comment.

## Negation

At first glance, negation seems redundant. Rather than searching for:

```
NOT ( summary   contains the string   "foo" )
```

one could search for:

```
summary   does not contain the string   "foo"
```

However, the search:

```
CC   does not contain the string   "@mozilla.org"
```

would find every bug where anyone on the CC list did not contain "@mozilla.org" while:

```
NOT ( CC   contains the string   "@mozilla.org" )
```

would find every bug where there was nobody on the CC list who did contain the string. Similarly, the use of negation also permits complex expressions to be built using terms OR'd together and then negated. Negation permits queries such as:

```
NOT ( ( product   equals   "Update" )
      OR
      ( component   equals   "Documentation" )
    )
```

to find bugs that are neither in the *Update* product or in the *Documentation* component or:

```
NOT ( ( commenter   equals   "%assignee%" )
      OR
      (component   equals   "Documentation" )
    )
```

to find non-documentation bugs on which the assignee has never commented.

### Pronoun Substitution

Sometimes, a query needs to compare a user-related field (such as *Reporter*) with a role-specific user (such as the user running the query or the user to whom each bug is assigned). For example, you may want to find all bugs that are assigned to the person who reported them.

When the *Custom Search* operator is either *equals* or *notequals*, the value can be `%reporter%`, `%triageowner%`, `%assignee%`, `%qacontact%`, `%user%` or `%self%`. These are known as "pronouns". The `%user%` pronoun and its alias `%self%` refer to the user who is executing the query (that's you) or, in the case of whining reports, the user who will be the recipient of the report. The `%reporter%`, `%triageowner%`, `%assignee%` and `%qacontact%` pronouns refer to the corresponding fields in the bug.

This feature also lets you search by a user's group memberships. If the operator is either *equals*, *notequals* or *anyexact*, you can search for whether a user belongs (or not) to the specified group. The group name must be entered using "%group.foo%" syntax, where "foo" is the group name. So if you are looking for bugs reported by any user being in the "editbugs" group, then you can use:

```
reporter   equals   "%group.editbugs%"
```

### Searching for Bugs Restricted to Groups

When administrators set up products, they can establish one or more groups bugs in the product can be associated with. If a bug is associated with a group then only users who are members of the group can see them.

This restriction is mostly used for security-related bugs, or internal tickets.

In order to search for bugs restricted to a group, you must be a member of the group.

Visit the Permissions page to find the groups you belong to, then search using the clause

> Group is equal to "%group.groupname%"

to list the bugs restricted to *groupname*.

### Searching on Relative Dates

In order to conduct searches over a window of time, you can use *relative dates* in query values.

The relative date values are of the form *nnV* where *nn* is a positive or negative integer and *V* is one of:

- *h* – for hours
- *d* – for days
- *w* – for weeks
- *m* – for months
- *y* – for years

A value of *1d* means 24 hours in the future from the time of the search.

A value of *-1d* means 24 hours in the past from the time of the search.

These relative values can be used when the *Custom Search* operator is one of:

- *is less than*
- *is less than or equal to*
- *is greater than*

- *is greater than or equal to*

and the field compared is a Datetime type.

To find bugs opened in the last 24 hours, you could search on:

> Opened is less than “-1d”

To find bugs opened during the current day (UTC),

> Opened is less than “-0ds”

Appending *s* to a relative date means *start of*.

You may also use relative dates for when a field changed. In the *Custom Search* operator that would be

- *changed after*
- *changed before*

To find bugs whose *priority* changed in the last seven days, search on:

> Priority changed after “-1w”

You can also search for a change to a particular value over a relative date using the *Search by Change History* operator.

To find the bugs *RESOLVED* as *WONTFIX* in the current year to date, you would search on

> Resolution changed to “WONTFIX” between “-0ys” and “NOW”

### 2.5.5 Bug Lists

The result of a search is a list of matching bugs.

The format of the list is configurable. For example, it can be sorted by clicking the column headings. Other useful features can be accessed using the links at the bottom of the list:

**Long Format:**
this gives you a large page with a non-editable summary of the fields of each bug.

**XML (icon):**
get the buglist in an XML format.

**CSV (icon):**
get the buglist as comma-separated values, for import into e.g. a spreadsheet.

**Feed (icon):**
get the buglist as an Atom feed. Copy this link into your favorite feed reader. If you are using Firefox, you can also save the list as a live bookmark by clicking the live bookmark icon in the status bar. To limit the number of bugs in the feed, add a limit=n parameter to the URL.

**iCalendar (icon):**
Get the buglist as an iCalendar file. Each bug is represented as a to-do item in the imported calendar.

**Change Columns:**
change the bug attributes which appear in the list.

**Change Several Bugs At Once:**
If your account is sufficiently empowered, and more than one bug appears in the bug list, this link is displayed and lets you easily make the same change to all the bugs in the list - for example, changing their assignee.

**Send Mail to Bug Assignees:**
If more than one bug appear in the bug list and there are at least two distinct bug assignees, this links is displayed which lets you easily send a mail to the assignees of all bugs on the list.

**Edit Search:**
If you didn't get exactly the results you were looking for, you can return to the Query page through this link and make small revisions to the query you just made so you get more accurate results.

**Remember Search As:**
You can give a search a name and remember it; the name will appear as an auto-completion in the search field in the header of Bugzilla pages giving you quick access to run it again later.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 2.6 Reports and Charts

As well as the standard buglist, Bugzilla has two more ways of viewing sets of bugs. These are the reports (which give different views of the current state of the database) and charts (which plot the changes in particular sets of bugs over time).

### 2.6.1 Reports

A report is a view of the current state of the bug database.

You can run either an HTML-table-based report, or a graphical line/pie/bar-chart-based one. The two have different pages to define them but are close cousins - once you've defined and viewed a report, you can switch between any of the different views of the data at will.

Both report types are based on the idea of defining a set of bugs using the standard search interface and then choosing some aspect of that set to plot on the horizontal and/or vertical axes. You can also get a form of 3-dimensional report by choosing to have multiple images or tables.

So, for example, you could use the search form to choose "all bugs in the WorldControl product" and then plot their severity against their component to see which component has had the largest number of bad bugs reported against it.

Once you've defined your parameters and hit *Generate Report*, you can switch between HTML, CSV, Bar, Line and Pie. (Note: Pie is only available if you didn't define a vertical axis, as pie charts don't have one.) The other controls are fairly self-explanatory; you can change the size of the image if you find text is overwriting other text, or the bars are too thin to see.

### 2.6.2 Charts

A chart is a view of the state of the bug database over time.

Bugzilla currently has two charting systems - Old Charts and New Charts. Old Charts have been part of Bugzilla for a long time; they chart each status and resolution for each product, and that's all. They are deprecated, and going away soon - we won't say any more about them. New Charts are the future - they allow you to chart anything you can define as a search.

---

**Note:** Both charting forms require the administrator to set up the data-gathering script. If you can't see any charts, ask them whether they have done so.

---

An individual line on a chart is called a data set. All data sets are organized into categories and subcategories. The data sets that Bugzilla defines automatically use the Product name as a *Category* and Component names as *Subcategories*, but there is no need for you to follow that naming scheme with your own charts if you don't want to.

Data sets may be public or private. Everyone sees public data sets in the list, but only their creator sees private data sets. Only administrators can make data sets public. No two data sets, even two private ones, can have the same set of category, subcategory and name. So if you are creating private data sets, one idea is to have the *Category* be your username.

#### Creating Charts

You create a chart by selecting a number of data sets from the list and pressing *Add To List* for each. In the *List Of Data Sets To Plot*, you can define the label that data set will have in the chart's legend and also ask Bugzilla to *Sum* a number of data sets (e.g. you could *Sum* data sets representing *RESOLVED*, *VERIFIED* and *CLOSED* in a particular product to get a data set representing all the resolved bugs in that product.)

If you've erroneously added a data set to the list, select it using the checkbox and click *Remove*. Once you add more than one data set, a *Grand Total* line automatically appears at the bottom of the list. If you don't want this, simply remove it as you would remove any other line.

You may also choose to plot only over a certain date range, and to cumulate the results, that is, to plot each one using the previous one as a baseline so the top line gives a sum of all the data sets. It's easier to try than to explain :-)

Once a data set is in the list, you can also perform certain actions on it. For example, you can edit the data set's parameters (name, frequency etc.) if it's one you created or if you are an administrator.

Once you are happy, click *Chart This List* to see the chart.

#### Creating New Data Sets

You may also create new data sets of your own. To do this, click the *create a new data set* link on the *Create Chart* page. This takes you to a search-like interface where you can define the search that Bugzilla will plot. At the bottom of the page, you choose the category, sub-category and name of your new data set.

If you have sufficient permissions, you can make the data set public, and reduce the frequency of data collection to less than the default of seven days.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 2.7 Pro Tips

This section distills some Bugzilla tips and best practices that have been developed.

### 2.7.1 Autolinkification

Bugzilla comments are plain text - so typing <U> will produce less-than, U, greater-than rather than underlined text. However, Bugzilla will automatically make hyperlinks out of certain sorts of text in comments. For example, the text `https://www.bugzilla.org` will be turned into a link: https://www.bugzilla.org. Other strings which get linkified in the obvious manner are:

- bug 12345
- bugs 123, 456, 789
- comment 7
- comments 1, 2, 3, 4

- bug 23456, comment 53
- attachment 4321
- mailto:george@example.com
- george@example.com
- ftp://ftp.mozilla.org
- Most other sorts of URL

A corollary here is that if you type a bug number in a comment, you should put the word “bug” before it, so it gets autolinkified for the convenience of others.

### 2.7.2 Comments

If you are changing the fields on a bug, only comment if either you have something pertinent to say or Bugzilla requires it. Otherwise, you may spam people unnecessarily with bugmail. To take an example: a user can set up their account to filter out messages where someone just adds themselves to the CC field of a bug (which happens a lot). If you come along, add yourself to the CC field, and add a comment saying “Adding self to CC”, then that person gets a pointless piece of mail they would otherwise have avoided.

Don’t use signs in comments. Signing your name (“Bill”) is acceptable, if you do it out of habit, but full mail/news-style four line ASCII art creations are not.

If you feel a bug you filed was incorrectly marked as a DUPLICATE of another, please question it in your bug, not the bug it was duped to. Feel free to CC the person who duped it if they are not already CCed.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 2.8 User Preferences

Once logged in, you can customize various aspects of Bugzilla via the “Preferences” link in the page footer. The preferences are split into a number of tabs, detailed in the sections below.

### 2.8.1 General Preferences

This tab allows you to change several default settings of Bugzilla. Administrators have the power to remove preferences from this list, so you may not see all the preferences available.

Each preference should be self-explanatory.

### 2.8.2 Email Preferences

This tab allows you to enable or disable email notification on specific events.

In general, users have almost complete control over how much (or how little) email Bugzilla sends them. If you want to receive the maximum amount of email possible, click the `Enable All Mail` button. If you don’t want to receive any email from Bugzilla at all, click the `Disable All Mail` button.

**Note:** A Bugzilla administrator can stop a user from receiving bugmail by clicking the `Bugmail Disabled` checkbox when editing the user account. This is a drastic step best taken only for disabled accounts, as it overrides the user's individual mail preferences.

There are two global options – `Email me when someone asks me to set a flag` and `Email me when someone sets a flag I asked for`. These define how you want to receive bugmail with regards to flags. Their use is quite straightforward: enable the checkboxes if you want Bugzilla to send you mail under either of the above conditions.

If you'd like to set your bugmail to something besides 'Completely ON' and 'Completely OFF', the `Field/recipient specific options` table allows you to do just that. The rows of the table define events that can happen to a bug – things like attachments being added, new comments being made, the priority changing, etc. The columns in the table define your relationship with the bug - reporter, assignee, QA contact (if enabled) or CC list member.

To fine-tune your bugmail, decide the events for which you want to receive bugmail; then decide if you want to receive it all the time (enable the checkbox for every column) or only when you have a certain relationship with a bug (enable the checkbox only for those columns). For example, if you didn't want to receive mail when someone added themselves to the CC list, you could uncheck all the boxes in the `CC Field Changes` line. As another example, if you never wanted to receive email on bugs you reported unless the bug was resolved, you would uncheck all boxes in the `Reporter` column except for the one on the `The bug is resolved or verified` row.

**Note:** Bugzilla adds the `X-Bugzilla-Reason` header to all bugmail it sends, describing the recipient's relationship (AssignedTo, Reporter, QAContact, CC, or Voter) to the bug. This header can be used to do further client-side filtering.

Bugzilla has a feature called `User Watching`. When you enter one or more comma-delineated user accounts (usually email addresses) into the text entry box, you will receive a copy of all the bugmail those users are sent (security settings permitting). This powerful functionality enables seamless transitions as developers change projects or users go on holiday.

Each user listed in the `Users watching you` field has you listed in their `Users to watch` list and can get bugmail according to your relationship to the bug and their `Field/recipient specific options` setting.

Lastly, you can define a list of bugs on which you no longer wish to receive any email, ever. (You can also add bugs to this list individually by checking the "Ignore Bug Mail" checkbox on the bug page for that bug.) This is useful for ignoring bugs where you are the reporter, as that's a role it's not possible to stop having.

### 2.8.3 Saved Searches

On this tab you can view and run any Saved Searches that you have created, and any Saved Searches that other members of the group defined in the querysharegroup parameter have shared. Saved Searches can be added to the page footer from this screen. If somebody is sharing a Search with a group they are allowed to *assign users to*, the sharer may opt to have the Search show up in the footer of the group's direct members by default.

### 2.8.4 Account Information

On this tab, you can change your basic account information, including your password, email address and real name. For security reasons, in order to change anything on this page you must type your *current* password into the `Password` field at the top of the page. If you attempt to change your email address, a confirmation email is sent to both the old and new addresses with a link to use to confirm the change. This helps to prevent account hijacking.

### 2.8.5 API Keys

API keys allow you to give a “token” to some external software so it can log in to the WebService API as you without knowing your password. You can then revoke that token if you stop using the web service, and you don’t need to change your password everywhere.

You can create more than one API key if required. Each API key has an optional description which can help you record what it is used for.

On this page, you can unrevoke, revoke, make sticky, and change the description of existing API keys for your login. A revoked key means that it cannot be used. The description is optional and purely for your information.

Sticky API keys may only be used from one IP address, which reduces the risk of the key being leaked. The IP address is the one the key was last used from. The expected workflow is that the sticky bit will be set once your application (or script) is setup. The sticky attribute may only be set, it can’t ever be unset.

You can also create a new API key by selecting the checkbox under the ‘New API key’ section of the page.

### 2.8.6 Permissions

This is a purely informative page which outlines your current permissions on this installation of Bugzilla.

A complete list of permissions in a default install of Bugzilla is below. Your administrator may have defined other permissions. Only users with *editusers* privileges can change the permissions of other users.

**admin**
Indicates user is an Administrator.

**bz_canusewhineatothers**
Indicates user can configure whine reports for other users.

**bz_canusewhines**
Indicates user can configure whine reports for self.

**bz_quip_moderators**
Indicates user can moderate quips.

**bz_sudoers**
Indicates user can perform actions as other users.

**bz_sudo_protect**
Indicates user cannot be impersonated by other users.

**canconfirm**
Indicates user can confirm a bug or mark it a duplicate.

**creategroups**
Indicates user can create and destroy groups.

**editbugs**
Indicates user can edit all bug fields.

**editclassifications**
: Indicates user can create, destroy and edit classifications.

**editcomponents**
: Indicates user can create, destroy and edit products, components, versions, milestones and flag types.

**editkeywords**
: Indicates user can create, destroy and edit keywords.

**editusers**
: Indicates user can create, disable and edit users.

**tweakparams**
: Indicates user can change *Parameters*.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 2.9 Installed Extensions

Bugzilla can be enhanced using extensions (see *Extensions*). If an extension comes with documentation in the appropriate format, and you build your own copy of the Bugzilla documentation using `makedocs.pl`, then the documentation for your installed extensions will show up here.

Your Bugzilla installation has the following extensions available (as of the last time you compiled the documentation):

---

This documentation undoubtedly has bugs; if you find some, please file them here.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

CHAPTER

THREE

# ADMINISTRATION GUIDE

For those with admin privileges, Bugzilla can be administered using the *Administration* link in the header. The administrative controls are divided into several sections:

## 3.1 Parameters

Bugzilla is configured by changing various parameters, accessed from the *Parameters* link, which is found on the Administration page. The parameters are divided into several categories, accessed via the menu on the left.

### 3.1.1 General

**maintainer**
: Email address of the person responsible for maintaining this Bugzilla installation. The address need not be that of a valid Bugzilla account.

**utf8**
: Use UTF-8 (Unicode) encoding for all text in Bugzilla. Installations where this parameter is set to off should set it to on only after the data has been converted from existing legacy character encodings to UTF-8, using the `contrib/recode.pl` script.

> **Note:** If you turn this parameter from off to on, you must re-run `checksetup.pl` immediately afterward.

**announcehtml**
: Any text in this field will be displayed at the top of every HTML page in this Bugzilla installation. The text is not wrapped in any tags. For best results, wrap the text in a <p> tag. Any style attributes from the CSS can be applied. `<p class="warning">` makes the text red.

**upgrade_notification**
: Enable or disable a notification on the homepage of this Bugzilla installation when a newer version of Bugzilla is available. This notification is only visible to administrators. Choose disabled to turn off the notification. Otherwise, choose which version of Bugzilla you want to be notified about: development_snapshot is the latest release from the master branch, latest_stable_release is the most recent release available on the most recent stable branch, and stable_branch_release is the most recent release on the branch this installation is based on.

### 3.1.2 Administrative Policies

This page contains parameters for basic administrative functions. Options include whether to allow the deletion of bugs and users, and whether to allow users to change their email address.

**allowbugdeletion**
The pages to edit products and components can delete all associated bugs when you delete a product (or component). Since that is a pretty scary idea, you have to turn on this option before any such deletions will ever happen.

**allowemailchange**
Users can change their own email address through the preferences. Note that the change is validated by emailing both addresses, so switching this option on will not let users use an invalid address.

**allowuserdeletion**
The user editing pages are capable of letting you delete user accounts. Bugzilla will issue a warning in case you'd run into inconsistencies when you're about to do so, but such deletions still remain scary. So, you have to turn on this option before any such deletions will ever happen.

**last_visit_keep_days**
This option controls how many days Bugzilla will remember that users have visited specific bugs.

### 3.1.3 User Authentication

This page contains the settings that control how this Bugzilla installation will do its authentication. Choose what authentication mechanism to use (the Bugzilla database, or an external source such as LDAP), and set basic behavioral parameters. For example, choose whether to require users to login to browse bugs, the management of authentication cookies, and the regular expression used to validate email addresses. Some parameters are highlighted below.

**auth_env_id**
Environment variable used by external authentication system to store a unique identifier for each user. Leave it blank if there isn't one or if this method of authentication is not being used.

**auth_env_email**
Environment variable used by external authentication system to store each user's email address. This is a required field for environmental authentication. Leave it blank if you are not going to use this feature.

**auth_env_realname**
Environment variable used by external authentication system to store the user's real name. Leave it blank if there isn't one or if this method of authentication is not being used.

**user_info_class**
Mechanism(s) to be used for gathering a user's login information. More than one may be selected. If the first one returns nothing, the second is tried, and so on. The types are:

- CGI: asks for username and password via CGI form interface.
- Env: info for a pre-authenticated user is passed in system environment variables.

**user_verify_class**
Mechanism(s) to be used for verifying (authenticating) information gathered by user_info_class. More than one may be selected. If the first one cannot find the user, the second is tried, and so on. The types are:

- DB: Bugzilla's built-in authentication. This is the most common choice.
- RADIUS: RADIUS authentication using a RADIUS server. Using this method requires additional parameters to be set. Please see *RADIUS* for more information.
- LDAP: LDAP authentication using an LDAP server. Using this method requires additional parameters to be set. Please see *LDAP* for more information.

**rememberlogin**
Controls management of session cookies.

- on - Session cookies never expire (the user has to login only once per browser).
- off - Session cookies last until the users session ends (the user will have to login in each new browser session).
- defaulton/defaultoff - Default behavior as described above, but user can choose whether Bugzilla will remember their login or not.

**requirelogin**
If this option is set, all access to the system beyond the front page will require a login. No anonymous users will be permitted.

**webservice_email_filter**
Filter email addresses returned by the WebService API depending on if the user is logged in or not. This works similarly to how the web UI currently filters email addresses. If requirelogin is enabled, then this parameter has no effect as users must be logged in to use Bugzilla anyway.

**emailregexp**
Defines the regular expression used to validate email addresses used for login names. The default attempts to match fully qualified email addresses (i.e. 'user@example.com') in a slightly more restrictive way than what is allowed in RFC 2822. Another popular value to put here is ^[^@]+, which means 'local usernames, no @ allowed.'

**emailregexpdesc**
This description is shown to the user to explain which email addresses are allowed by the emailregexp param.

**emailsuffix**
This is a string to append to any email addresses when actually sending mail to that address. It is useful if you have changed the emailregexp param to only allow local usernames, but you want the mail to be delivered to username@my.local.hostname.

**createemailregexp**
This defines the (case-insensitive) regexp to use for email addresses that are permitted to self-register. The default (.*) permits any account matching the emailregexp to be created. If this parameter is left blank, no users will be permitted to create their own accounts and all accounts will have to be created by an administrator.

**password_check_on_login**
If set, Bugzilla will check that the password meets the current complexity rules and minimum length requirements when the user logs into the Bugzilla web interface. If it doesn't, the user would not be able to log in, and will receive a message to reset their password.

**auth_delegation**
If set, Bugzilla will allow other websites to request API keys from its own users. See *Authentication Delegation via API Keys*.

### 3.1.4 Attachments

This page allows for setting restrictions and other parameters regarding attachments to bugs. For example, control size limitations and whether to allow pointing to external files via a URI.

**allow_attachment_display**
If this option is on, users will be able to view attachments from their browser, if their browser supports the attachment's MIME type. If this option is off, users are forced to download attachments, even if the browser is able to display them.

If you do not trust your users (e.g. if your Bugzilla is public), you should either leave this option off, or configure and set the attachment_base localconfig variable. Untrusted users may upload attachments that could be potentially damaging if viewed directly in the browser.

**allow_attachment_deletion**
If this option is on, administrators will be able to delete the contents of attachments (i.e. replace the attached file with a 0 byte file), leaving only the metadata.

**maxattachmentsize**
The maximum size (in kilobytes) of attachments to be stored in the database. If a file larger than this size is attached to a bug, Bugzilla will look at the maxlocalattachment parameter to determine if the file can be stored locally on the web server. If the file size exceeds both limits, then the attachment is rejected. Setting both parameters to 0 will prevent attaching files to bugs.

Some databases have default limits which prevent storing larger attachments in the database. E.g. MySQL has a parameter called max_allowed_packet, whose default varies by distribution. Setting maxattachmentsize higher than your current setting for this value will produce an error.

**maxlocalattachment**
The maximum size (in megabytes) of attachments to be stored locally on the web server. If set to a value lower than the maxattachmentsize parameter, attachments will never be kept on the local filesystem.

Whether you use this feature or not depends on your environment. Reasons to store some or all attachments as files might include poor database performance for large binary blobs, ease of backup/restore/browsing, or even filesystem-level deduplication support. However, you need to be aware of any limits on how much data your webserver environment can store. If in doubt, leave the value at 0.

Note that changing this value does not affect any already-submitted attachments.

### 3.1.5 Bug Change Policies

Set policy on default behavior for bug change events. For example, choose which status to set a bug to when it is marked as a duplicate, and choose whether to allow bug reporters to set the priority or target milestone. Also allows for configuration of what changes should require the user to make a comment, described below.

**duplicate_or_move_bug_status**
When a bug is marked as a duplicate of another one, use this bug status.

**letsubmitterchoosepriority**
If this is on, then people submitting bugs can choose an initial priority for that bug. If off, then all bugs initially have the default priority selected here.

**letsubmitterchoosemilestone**
If this is on, then people submitting bugs can choose the Target Milestone for that bug. If off, then all bugs initially have the default milestone for the product being filed in.

**musthavemilestoneonaccept**
If you are using Target Milestone, do you want to require that the milestone be set in order for a user to set a bug's status to IN_PROGRESS?

**commenton***
All these fields allow you to dictate what changes can pass without comment and which must have a comment from the person who changed them. Often, administrators will allow users to add themselves to the CC list, accept bugs, or change the Status Whiteboard without adding a comment as to their reasons for the change, yet require that most other changes come with an explanation. Set the "commenton" options according to your site policy. It is a wise idea to require comments when users resolve, reassign, or reopen bugs at the very least.

---

**Note:** It is generally far better to require a developer comment when resolving bugs than not. Few things are more annoying to bug database users than having a developer mark a bug "fixed" without any comment as to what the fix was (or even that it was truly fixed!)

**noresolveonopenblockers**
This option will prevent users from resolving bugs as FIXED if they have unresolved dependencies. Only the FIXED resolution is affected. Users will be still able to resolve bugs to resolutions other than FIXED if they have unresolved dependent bugs.

### 3.1.6 Bug Fields

The parameters in this section determine the default settings of several Bugzilla fields for new bugs and whether certain fields are used. For example, choose whether to use the Target Milestone field or the Status Whiteboard field.

**useclassification**
If this is on, Bugzilla will associate each product with a specific classification. But you must have editclassification permissions enabled in order to edit classifications.

**usetargetmilestone**
Do you wish to use the Target Milestone field?

**useqacontact**
This allows you to define an email address for each component, in addition to that of the default assignee, that will be sent carbon copies of incoming bugs.

**usestatuswhiteboard**
This defines whether you wish to have a free-form, overwritable field associated with each bug. The advantage of the Status Whiteboard is that it can be deleted or modified with ease and provides an easily searchable field for indexing bugs that have some trait in common.

**use_regression_fields**
Do you wish to use the Regressions and Regressed by fields? These allow you to efficiently track software regressions, which might previously be managed using the Depends on and Blocks fields along with the "regression" keyword.

**use_see_also**
Do you wish to use the See Also field? It allows you mark bugs in other bug tracker installations as being related. Disabling this field prevents addition of new relationships, but existing ones will continue to appear.

**require_bug_type**
If this is on, users are asked to choose a type when they file a new bug.

**default_bug_type**
This is the type that newly entered bugs are set to.

**defaultpriority**
This is the priority that newly entered bugs are set to.

**defaultseverity**
This is the severity that newly entered bugs are set to.

**defaultplatform**
This is the platform that is preselected on the bug entry form. You can leave this empty; Bugzilla will then use the platform that the browser is running on as the default.

**defaultopsys**
This is the operating system that is preselected on the bug entry form. You can leave this empty; Bugzilla will then use the operating system that the browser reports to be running on as the default.

**collapsed_comment_tags**
A comma-separated list of tags which, when applied to comments, will cause them to be collapsed by default.

### 3.1.7 Graphs

Bugzilla can draw graphs of bug-dependency relationships, using a tool called `dot` (from the GraphViz project) or a web service called Web Dot. This page allows you to set the location of the binary or service. If no Web Dot server or binary is specified, then dependency graphs will be disabled.

**webdotbase**
You may set this parameter to any of the following:

- A complete file path to **`dot`** (part of GraphViz), which will generate the graphs locally.
- A URL prefix pointing to an installation of the Web Dot package, which will generate the graphs remotely.
- A blank value, which will disable dependency graphing.

The default value is blank. We recommend using a local install of `dot`. If you change this value to a web service, make certain that the Web Dot server can read files from your Web Dot directory. On Apache you do this by editing the `.htaccess` file; for other systems the needed measures may vary. You can run **`checksetup.pl`** to recreate the `.htaccess` file if it has been lost.

**font_file**
You can specify the full path to a TrueType font file which will be used to display text (labels, legends, ...) in charts and graphical reports. To support as many languages as possible, we recommend to specify a TrueType font such as Unifont which supports all printable characters in the Basic Multilingual Plane. If you leave this parameter empty, a default font will be used, but its support is limited to English characters only and so other characters will be displayed incorrectly.

### 3.1.8 Group Security

Bugzilla allows for the creation of different groups, with the ability to restrict the visibility of bugs in a group to a set of specific users. Specific products can also be associated with groups, and users restricted to only see products in their groups. Several parameters are described in more detail below. Most of the configuration of groups and their relationship to products is done on the *Groups* and *Product* pages of the *Administration* area. The options on this page control global default behavior. For more information on Groups and Group Security, see *Groups and Security*.

**makeproductgroups**
Determines whether or not to automatically create groups when new products are created. If this is on, the groups will be used for querying bugs.

**chartgroup**
The name of the group of users who can use the ‘New Charts’ feature. Administrators should ensure that the public categories and series definitions do not divulge confidential information before enabling this for an untrusted population. If left blank, no users will be able to use New Charts.

**insidergroup**
The name of the group of users who can see/change private comments and attachments.

**timetrackinggroup**
The name of the group of users who can see/change time tracking information.

**querysharegroup**
The name of the group of users who are allowed to share saved searches with one another. For more information on using saved searches, see *Saved Searches*.

**comment_taggers_group**
The name of the group of users who can tag comments. Setting this to empty disables comment tagging.

**debug_group**
The name of the group of users who can view the actual SQL query generated when viewing bug lists and reports. Do not expose this information to untrusted users.

**usevisibilitygroups**
If selected, user visibility will be restricted to members of groups, as selected in the group configuration settings. Each user-defined group can be allowed to see members of selected other groups. For details on configuring groups (including the visibility restrictions) see *Editing Groups and Assigning Group Permissions*.

**or_groups**
Define the visibility of a bug which is in multiple groups. If this is on (recommended), a user only needs to be a member of one of the bug's groups in order to view it. If it is off, a user needs to be a member of all the bug's groups. Note that in either case, a user's role on the bug (e.g. reporter), if any, may also affect their permissions.

### 3.1.9 LDAP

LDAP authentication is a module for Bugzilla's plugin authentication architecture. This page contains all the parameters necessary to configure Bugzilla for use with LDAP authentication.

The existing authentication scheme for Bugzilla uses email addresses as the primary user ID and a password to authenticate that user. All places within Bugzilla that require a user ID (e.g. assigning a bug) use the email address. The LDAP authentication builds on top of this scheme, rather than replacing it. The initial log-in is done with a username and password for the LDAP directory. Bugzilla tries to bind to LDAP using those credentials and, if successful, tries to map this account to a Bugzilla account. If an LDAP mail attribute is defined, the value of this attribute is used; otherwise, the emailsuffix parameter is appended to the LDAP username to form a full email address. If an account for this address already exists in the Bugzilla installation, it will log in to that account. If no account for that email address exists, one is created at the time of login. (In this case, Bugzilla will attempt to use the "displayName" or "cn" attribute to determine the user's full name.) After authentication, all other user-related tasks are still handled by email address, not LDAP username. For example, bugs are still assigned by email address and users are still queried by email address.

**Warning:** Because the Bugzilla account is not created until the first time a user logs in, a user who has not yet logged is unknown to Bugzilla. This means they cannot be used as an assignee or QA contact (default or otherwise), added to any CC list, or any other such operation. One possible workaround is the `bugzilla_ldapsync.rb` script in the `contrib` directory. Another possible solution is fixing bug 201069.

Parameters required to use LDAP Authentication:

**user_verify_class (in the Authentication section)**
If you want to list LDAP here, make sure to have set up the other parameters listed below. Unless you have other (working) authentication methods listed as well, you may otherwise not be able to log back in to Bugzilla once you log out. If this happens to you, you will need to manually edit `data/params.json` and set user_verify_class to DB.

**LDAPserver**
This parameter should be set to the name (and optionally the port) of your LDAP server. If no port is specified, it assumes the default LDAP port of 389. For example: ldap.company.com or ldap.company.com:3268 You can also specify a LDAP URI, so as to use other protocols, such as LDAPS or LDAPI. If the port was not specified in the URI, the default is either 389 or 636 for 'LDAP' and 'LDAPS' schemes respectively.

**Note:** In order to use SSL with LDAP, specify a URI with "ldaps://". This will force the use of SSL over port 636. For example, normal LDAP ldap://ldap.company.com, LDAP over SSL ldaps://ldap.company.com, or

LDAP over a UNIX domain socket ldapi://%2fvar%2flib%2fldap_sock.

**LDAPstarttls**
Whether to require encrypted communication once a normal LDAP connection is achieved with the server.

**LDAPbinddn [Optional]**
Some LDAP servers will not allow an anonymous bind to search the directory. If this is the case with your configuration you should set the LDAPbinddn parameter to the user account Bugzilla should use instead of the anonymous bind. Ex. cn=default,cn=user:password

**LDAPBaseDN**
The location in your LDAP tree that you would like to search for email addresses. Your uids should be unique under the DN specified here. Ex. ou=People,o=Company

**LDAPuidattribute**
The attribute which contains the unique UID of your users. The value retrieved from this attribute will be used when attempting to bind as the user to confirm their password. Ex. uid

**LDAPmailattribute**
The name of the attribute which contains the email address your users will enter into the Bugzilla login boxes. Ex. mail

**LDAPfilter**
LDAP filter to AND with the LDAPuidattribute for filtering the list of valid users.

### 3.1.10 RADIUS

RADIUS authentication is a module for Bugzilla's plugin authentication architecture. This page contains all the parameters necessary for configuring Bugzilla to use RADIUS authentication.

**Note:** Most caveats that apply to LDAP authentication apply to RADIUS authentication as well. See *LDAP* for details.

Parameters required to use RADIUS Authentication:

**user_verify_class (in the Authentication section)**
If you want to list RADIUS here, make sure to have set up the other parameters listed below. Unless you have other (working) authentication methods listed as well, you may otherwise not be able to log back in to Bugzilla once you log out. If this happens to you, you will need to manually edit `data/params.json` and set user_verify_class to DB.

**RADIUS_server**
The name (and optionally the port) of your RADIUS server.

**RADIUS_secret**
The RADIUS server's secret.

**RADIUS_NAS_IP**
The NAS-IP-Address attribute to be used when exchanging data with your RADIUS server. If unspecified, 127.0.0.1 will be used.

**RADIUS_email_suffix**
Bugzilla needs an email address for each user account. Therefore, it needs to determine the email address corresponding to a RADIUS user. Bugzilla offers only a simple way to do this: it can concatenate a suffix to the RADIUS user name to convert it into an email address. You can specify this suffix in the RADIUS_email_suffix parameter. If this simple solution does not work for you, you'll probably need to modify `Bugzilla/Auth/Verify/RADIUS.pm` to match your requirements.

### 3.1.11 Email

This page contains all of the parameters for configuring how Bugzilla deals with the email notifications it sends. See below for a summary of important options.

**mail_delivery_method**
This is used to specify how email is sent, or if it is sent at all. There are several options included for different MTAs, along with two additional options that disable email sending. Test does not send mail, but instead saves it in `data/mailer.testfile` for later review. None disables email sending entirely.

**mailfrom**
This is the email address that will appear in the "From" field of all emails sent by this Bugzilla installation. Some email servers require mail to be from a valid email address; therefore, it is recommended to choose a valid email address here.

**use_mailer_queue**
In a large Bugzilla installation, updating bugs can be very slow because Bugzilla sends all email at once. If you enable this parameter, Bugzilla will queue all mail and then send it in the background. This requires that you have installed certain Perl modules (as listed by `checksetup.pl` for this feature), and that you are running the `jobqueue.pl` daemon (otherwise your mail won't get sent). This affects all mail sent by Bugzilla, not just bug updates.

**smtpserver**
The SMTP server address, if the mail_delivery_method parameter is set to SMTP. Use localhost if you have a local MTA running; otherwise, use a remote SMTP server. Append ":" and the port number if a non-default port is needed.

**smtp_username**
Username to use for SASL authentication to the SMTP server. Leave this parameter empty if your server does not require authentication.

**smtp_password**
Password to use for SASL authentication to the SMTP server. This parameter will be ignored if the smtp_username parameter is left empty.

**smtp_ssl**
Enable SSL support for connection to the SMTP server.

**smtp_debug**
This parameter allows you to enable detailed debugging output. Log messages are printed the web server's error log.

**whinedays**
Set this to the number of days you want to let bugs go in the CONFIRMED state before notifying people they have untouched new bugs. If you do not plan to use this feature, simply do not set up the whining cron job described in the installation instructions, or set this value to "0" (never whine).

**globalwatchers**
This allows you to define specific users who will receive notification each time any new bug in entered, or when any existing bug changes, subject to the normal groupset permissions. It may be useful for sending notifications to a mailing list, for instance.

### 3.1.12 Query Defaults

This page controls the default behavior of Bugzilla in regards to several aspects of querying bugs. Options include what the default query options are, what the "My Bugs" page returns, whether users can freely add bugs to the quip list, and how many duplicate bugs are needed to add a bug to the "most frequently reported" list.

**quip_list_entry_control**
Controls how easily users can add entries to the quip list.

- open - Users may freely add to the quip list, and their entries will immediately be available for viewing.
- moderated - Quips can be entered but need to be approved by a moderator before they will be shown.
- closed - No new additions to the quips list are allowed.

**mybugstemplate**
This is the URL to use to bring up a simple 'all of my bugs' list for a user. %userid% will get replaced with the login name of a user. Special characters must be URL encoded.

**defaultquery**
This is the default query that initially comes up when you access the advanced query page. It's in URL-parameter format.

**search_allow_no_criteria**
When turned off, a query must have some criteria specified to limit the number of bugs returned to the user. When turned on, a user is allowed to run a query with no criteria and get all bugs in the entire installation that they can see. Turning this parameter on is not recommended on large installations.

**default_search_limit**
By default, Bugzilla limits searches done in the web interface to returning only this many results, for performance reasons. (This only affects the HTML format of search results—CSV, XML, and other formats are exempted.) Users can click a link on the search result page to see all the results.

Usually you should not have to change this—the default value should be acceptable for most installations.

**max_search_results**
The maximum number of bugs that a search can ever return. Tabular and graphical reports are exempted from this limit, however.

### 3.1.13 Shadow Database

This page controls whether a shadow database is used. If your Bugzilla is not large, you will not need these options.

A standard large database setup involves a single master server and a pool of read-only slaves (which Bugzilla calls the "shadowdb"). Queries which are not updating data can be directed to the slave pool, removing the load/locking from the master, freeing it up to handle writes. Bugzilla will switch to the shadowdb when it knows it doesn't need to update the database (e.g. when searching, or displaying a bug to a not-logged-in user).

Bugzilla does not make sure the shadowdb is kept up to date, so, if you use one, you will need to set up replication in your database server.

If your shadowdb is on a different machine, specify shadowdbhost and shadowdbport. If it's on the same machine, specify shadowdbsock.

**shadowdbhost**
The host the shadow database is on.

**shadowdbport**
The port the shadow database is on.

**shadowdbsock**
The socket used to connect to the shadow database, if the host is the local machine.

**shadowdb**
The database name of the shadow database.

### 3.1.14 User Matching

The settings on this page control how users are selected and queried when adding a user to a bug. For example, users need to be selected when assigning the bug, adding to the CC list, or selecting a QA contact. With the usemenuforusers parameter, it is possible to configure Bugzilla to display a list of users in the fields instead of an empty text field. If users are selected via a text box, this page also contains parameters for how user names can be queried and matched when entered.

**usemenuforusers**
If this option is set, Bugzilla will offer you a list to select from (instead of a text entry field) where a user needs to be selected. This option should not be enabled on sites where there are a large number of users.

**ajax_user_autocompletion**
If this option is set, typing characters in a certain user fields will display a list of matches that can be selected from. It is recommended to only turn this on if you are using mod_perl; otherwise, the response will be irritatingly slow.

**maxusermatches**
Provide no more than this many matches when a user is searched for. If set to '1', no users will be displayed on ambiguous matches. This is useful for user-privacy purposes. A value of zero means no limit.

**confirmuniqueusermatch**
Whether a confirmation screen should be displayed when only one user matches a search entry.

### 3.1.15 Advanced

**inbound_proxies**
When inbound traffic to Bugzilla goes through a proxy, Bugzilla thinks that the IP address of the proxy is the IP address of every single user. If you enter a comma-separated list of IPs in this parameter, then Bugzilla will trust any `X-Forwarded-For` header sent from those IPs, and use the value of that header as the end user's IP address.

**proxy_url**
If this Bugzilla installation is behind a proxy, enter the proxy information here to enable Bugzilla to access the Internet. Bugzilla requires Internet access to utilize the upgrade_notification parameter. If the proxy requires authentication, use the syntax: http://user:pass@proxy_url/.

**strict_transport_security**
Enables the sending of the Strict-Transport-Security header along with HTTP responses on SSL connections. This adds greater security to your SSL connections by forcing the browser to always access your domain over SSL and never accept an invalid certificate. However, it should only be used if you have the ssl_redirect parameter turned on, Bugzilla is the only thing running on its domain (i.e., your urlbase is something like http://bugzilla.example.com/), and you never plan to stop supporting SSL.

- off - Don't send the Strict-Transport-Security header with requests.
- this_domain_only - Send the Strict-Transport-Security header with all requests, but only support it for the current domain.
- include_subdomains - Send the Strict-Transport-Security header along with the includeSubDomains flag, which will apply the security change to all subdomains. This is especially useful when combined with an attachment_base that exists as (a) subdomain(s) under the main Bugzilla domain.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 3.2 Default Preferences

Each user of Bugzilla can set certain preferences about how they want Bugzilla to behave. Here, you can say whether or not each of the possible preferences is available to the user and, if it is, what the default value is.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 3.3 Users

### 3.3.1 Creating Admin Users

When you first run checksetup.pl after installing Bugzilla, it will prompt you for the username (email address) and password for the first admin user. If for some reason you delete all the admin users, re-running checksetup.pl will again prompt you for a username and password and make a new admin.

If you wish to add more administrative users, add them to the “admin” group.

### 3.3.2 Searching For Users

If you have `editusers` privileges or if you are allowed to grant privileges for some groups, the *Users* link will appear in the Administration page.

The first screen is a search form to search for existing user accounts. You can run searches based either on the user ID, real name or login name (i.e. the email address, or just the first part of the email address if the emailsuffix parameter is set). The search can be conducted in different ways using the listbox to the right of the text entry box. You can match by case-insensitive substring (the default), regular expression, a *reverse* regular expression match (which finds every user name which does NOT match the regular expression), or the exact string if you know exactly who you are looking for. The search can be restricted to users who are in a specific group. By default, the restriction is turned off.

The search returns a list of users matching your criteria. User properties can be edited by clicking the login name. The Account History of a user can be viewed by clicking the “View” link in the Account History column. The Account History displays changes that have been made to the user account, the time of the change and the user who made the change. For example, the Account History page will display details of when a user was added or removed from a group.

### 3.3.3 Modifying Users

Once you have found your user, you can change the following fields:

- *Login Name*: This is generally the user’s full email address. However, if you have are using the emailsuffix parameter, this may just be the user’s login name. Unless you turn off the allowemailchange parameter, users can change their login names themselves (to any valid email address).
- *Real Name*: The user’s real name. Note that Bugzilla does not require this to create an account.
- *Password*: You can change the user’s password here. Users can automatically request a new password, so you shouldn’t need to do this often. If you want to disable an account, see Disable Text below.

- *Bugmail Disabled*: Mark this checkbox to disable bugmail and whinemail completely for this account. This checkbox replaces the data/nomail file which existed in older versions of Bugzilla.
- *Disable Text*: If you type anything in this box, including just a space, the user is prevented from logging in and from making any changes to bugs via the web interface. The HTML you type in this box is presented to the user when they attempt to perform these actions and should explain why the account was disabled. Users with disabled accounts will continue to receive mail from Bugzilla; furthermore, they will not be able to log in themselves to change their own preferences and stop it. If you want an account (disabled or active) to stop receiving mail, simply check the `Bugmail Disabled` checkbox above.

**Note:** Even users whose accounts have been disabled can still submit bugs via the email gateway, if one exists. The email gateway should *not* be enabled for secure installations of Bugzilla.

**Warning:** Don't disable all the administrator accounts!

- *<groupname>*: If you have created some groups, e.g. "securitysensitive", then checkboxes will appear here to allow you to add users to, or remove them from, these groups. The first checkbox gives the user the ability to add and remove other users as members of this group. The second checkbox adds the user themselves as a member of the group.
- *canconfirm*: This field is only used if you have enabled the "unconfirmed" status. If you enable this for a user, that user can then move bugs from "Unconfirmed" to a "Confirmed" status (e.g.: "New" status).
- *creategroups*: This option will allow a user to create and destroy groups in Bugzilla.
- *editbugs*: Unless a user has this bit set, they can only edit those bugs for which they are the assignee or the reporter. Even if this option is unchecked, users can still add comments to bugs.
- *editcomponents*: This flag allows a user to create new products and components, modify existing products and components, and destroy those that have no bugs associated with them. If a product or component has bugs associated with it, those bugs must be moved to a different product or component before Bugzilla will allow them to be destroyed.
- *editkeywords*: If you use Bugzilla's keyword functionality, enabling this feature allows a user to create and destroy keywords. A keyword must be removed from any bugs upon which it is currently set before it can be destroyed.
- *editusers*: This flag allows a user to do what you're doing right now: edit other users. This will allow those with the right to do so to remove administrator privileges from other users or grant them to themselves. Enable with care.
- *tweakparams*: This flag allows a user to change Bugzilla's Params (using `editparams.cgi`.)
- *<productname>*: This allows an administrator to specify the products in which a user can see bugs. If you turn on the makeproductgroups parameter in the Group Security Panel in the Parameters page, then Bugzilla creates one group per product (at the time you create the product), and this group has exactly the same name as the product itself. Note that for products that already exist when the parameter is turned on, the corresponding group will not be created. The user must still have the editbugs privilege to edit bugs in these products.

### 3.3.4 Creating New Users

#### Self-Registration

By default, users can create their own user accounts by clicking the `New Account` link at the bottom of each page (assuming they aren't logged in as someone else already). If you want to disable this self-registration, or if you want to restrict who can create their own user account, you have to edit the createemailregexp parameter in the `Configuration` page; see *Parameters*.

#### Administrator Registration

Users with `editusers` privileges, such as administrators, can create user accounts for other users:

1. After logging in, click the "Users" link at the footer of the query page, and then click "Add a new user".
2. Fill out the form presented. This page is self-explanatory. When done, click "Submit".

**Note:** Adding a user this way will *not* send an email informing them of their username and password. While useful for creating dummy accounts (watchers which shuttle mail to another system, for instance, or email addresses which are a mailing list), in general it is preferable to log out and use the `New Account` button to create users, as it will pre-populate all the required fields and also notify the user of their account name and password.

### 3.3.5 Deleting Users

If the allowuserdeletion parameter is turned on (see *Parameters*) then you can also delete user accounts. Note that, most of the time, this is not the best thing to do. If only a warning in a yellow box is displayed, then the deletion is safe. If a warning is also displayed in a red box, then you should NOT try to delete the user account, else you will get referential integrity problems in your database, which can lead to unexpected behavior, such as bugs not appearing in bug lists anymore, or data displaying incorrectly. You have been warned!

### 3.3.6 Impersonating Users

There may be times when an administrator would like to do something as another user. The **`sudo`** feature may be used to do this.

**Note:** To use the sudo feature, you must be in the *bz_sudoers* group. By default, all administrators are in this group.

If you have access to this feature, you may start a session by going to the Edit Users page, Searching for a user and clicking on their login. You should see a link below their login name titled "Impersonate this user". Click on the link. This will take you to a page where you will see a description of the feature and instructions for using it. After reading the text, simply enter the login of the user you would like to impersonate, provide a short message explaining why you are doing this, and press the button.

As long as you are using this feature, everything you do will be done as if you were logged in as the user you are impersonating.

**Warning:** The user you are impersonating will not be told about what you are doing. If you do anything that results in mail being sent, that mail will appear to be from the user you are impersonating. You should be extremely careful while using this feature.

This documentation undoubtedly has bugs; if you find some, please file them here.

## 3.4 Classifications, Products, Components, Versions, and Milestones

Bugs in Bugzilla are classified into one of a set of admin-defined Components. Components are themselves each part of a single Product. Optionally, Products can be part of a single Classification, adding a third level to the hierarchy.

### 3.4.1 Classifications

Classifications are used to group several related products into one distinct entity.

For example, if a company makes computer games, they could have a classification of "Games", and a separate product for each game. This company might also have a `Common` classification, containing products representing units of technology used in multiple games, and perhaps an `Other` classification containing a few special products that represent items that are not actually shipping products (for example, "Website", or "Administration").

The classifications layer is disabled by default; it can be turned on or off using the useclassification parameter in the *Bug Fields* section of *Parameters*.

Access to the administration of classifications is controlled using the *editclassifications* system group, which defines a privilege for creating, destroying, and editing classifications.

When activated, classifications will introduce an additional step when filling bugs (dedicated to classification selection), and they will also appear in the advanced search form.

### 3.4.2 Products

Products usually represent real-world shipping products. Many of Bugzilla's settings are configurable on a per-product basis.

When creating or editing products the following options are available:

**Product**
    The name of the product

**Description**
    A brief description of the product

**Open for bug entry**
    Deselect this box to prevent new bugs from being entered against this product.

**Enable the UNCONFIRMED status in this product**
    Select this option if you want to use the UNCONFIRMED status (see *Workflow*)

**Default milestone**
    Select the default milestone for this product.

**Version**
    Specify the default version for this product.

**Create chart datasets for this product**
    Select to make chart datasets available for this product.

It is compulsory to create at least one *component* in a product, and so you will be asked for the details of that too.

When editing a product you can change all of the above, and there is also a link to edit Group Access Controls; see *Assigning Group Controls to Products*.

### Creating New Products

To create a new product:

1. Select `Administration` from the footer and then choose `Products` from the main administration page.
2. Select the `Add` link in the bottom right.
3. Enter the details as outlined above.

### Editing Products

To edit an existing product, click the "Products" link from the "Administration" page. If the useclassification parameter is turned on, a table of existing classifications is displayed, including an "Unclassified" category. The table indicates how many products are in each classification. Click on the classification name to see its products. If the useclassification parameter is not in use, the table lists all products directly. The product table summarizes the information defined when the product was created. Click on the product name to edit these properties, and to access links to other product attributes such as the product's components, versions, milestones, and group access controls.

### Adding or Editing Components, Versions and Target Milestones

To add new or edit existing Components, Versions, or Target Milestones to a Product, select the "Edit Components", "Edit Versions", or "Edit Milestones" links from the "Edit Product" page. A table of existing Components, Versions, or Milestones is displayed. Click on an item name to edit the properties of that item. Below the table is a link to add a new Component, Version, or Milestone.

For more information on components, see *Components*.

For more information on versions, see *Versions*.

For more information on milestones, see *Milestones*.

### Assigning Group Controls to Products

On the `Edit Product` page, there is a link called `Edit Group Access Controls`. The settings on this page control the relationship of the groups to the product being edited.

Group Access Controls are an important aspect of using groups for isolating products and restricting access to bugs filed against those products. For more information on groups, including how to create, edit, add users to, and alter permission of, see *Groups and Security*.

After selecting the "Edit Group Access Controls" link from the "Edit Product" page, a table containing all user-defined groups for this Bugzilla installation is displayed. The system groups that are created when Bugzilla is installed are not applicable to Group Access Controls. Below is description of what each of these fields means.

Groups may be applicable (i.e. bugs in this product can be associated with this group), default (i.e. bugs in this product are in this group by default), and mandatory (i.e. bugs in this product must be associated with this group) for each product. Groups can also control access to bugs for a given product, or be used to make bugs for a product totally read-only unless the group restrictions are met. The best way to understand these relationships is by example. See *Common Applications of Group Controls* for examples of product and group relationships.

**Note:** Products and Groups are not limited to a one-to-one relationship. Multiple groups can be associated with the same product, and groups can be associated with more than one product.

If any group has *Entry* selected, then the product will restrict bug entry to only those users who are members of *all* the groups with *Entry* selected.

If any group has *Canedit* selected, then the product will be read-only for any users who are not members of *all* of the groups with *Canedit* selected. *Only* users who are members of all the *Canedit* groups will be able to edit bugs for this product. This is an additional restriction that enables finer-grained control over products rather than just all-or-nothing access levels.

The following settings let you choose privileges on a *per-product basis*. This is a convenient way to give privileges to some users for some products only, without having to give them global privileges which would affect all products.

Any group having *editcomponents* selected allows users who are in this group to edit all aspects of this product, including components, milestones, and versions.

Any group having *canconfirm* selected allows users who are in this group to confirm bugs in this product.

Any group having *editbugs* selected allows users who are in this group to edit all fields of bugs in this product.

The *MemberControl* and *OtherControl* are used in tandem to determine which bugs will be placed in this group. The only allowable combinations of these two parameters are listed in a table on the "Edit Group Access Controls" page. Consult this table for details on how these fields can be used. Examples of different uses are described below.

### Common Applications of Group Controls

The use of groups is best explained by providing examples that illustrate configurations for common use cases. The examples follow a common syntax: *Group: Entry, MemberControl, OtherControl, CanEdit, EditComponents, CanConfirm, EditBugs*, where "Group" is the name of the group being edited for this product. The other fields all correspond to the table on the "Edit Group Access Controls" page. If any of these options are not listed, it means they are not checked.

### Basic Product/Group Restriction

Suppose there is a product called "Bar". You would like to make it so that only users in the group "Foo" can enter bugs in the "Bar" product. Additionally, bugs filed in product "Bar" must be visible only to users in "Foo" (plus, by default, the reporter, assignee, and CC list of each bug) at all times. Furthermore, only members of group "Foo" should be able to edit bugs filed against product "Bar", even if other users could see the bug. This arrangement would achieved by the following:

```
Product Bar:
foo: ENTRY, MANDATORY/MANDATORY, CANEDIT
```

Perhaps such strict restrictions are not needed for product "Bar". Instead, you would like to make it so that only members of group "Foo" can enter bugs in product "Bar", but bugs in "Bar" are not required to be restricted in visibility to people in "Foo". Anyone with permission to edit a particular bug in product "Bar" can put the bug in group "Foo", even if they themselves are not in "Foo".

Furthermore, anyone in group "Foo" can edit all aspects of the components of product "Bar", can confirm bugs in product "Bar", and can edit all fields of any bug in product "Bar". That would be done like this:

```
Product Bar:
foo: ENTRY, SHOWN/SHOWN, EDITCOMPONENTS, CANCONFIRM, EDITBUGS
```

### General User Access With Security Group

To permit any user to file bugs against "Product A", and to permit any user to submit those bugs into a group called "Security":

```
Product A:
security: SHOWN/SHOWN
```

### General User Access With A Security Product

To permit any user to file bugs against product called "Security" while keeping those bugs from becoming visible to anyone outside the group "SecurityWorkers" (unless a member of the "SecurityWorkers" group removes that restriction):

```
Product Security:
securityworkers: DEFAULT/MANDATORY
```

### Product Isolation With a Common Group

To permit users of "Product A" to access the bugs for "Product A", users of "Product B" to access the bugs for "Product B", and support staff, who are members of the "Support Group" to access both, three groups are needed:

1. Support Group: Contains members of the support staff.
2. AccessA Group: Contains users of product A and the Support group.
3. AccessB Group: Contains users of product B and the Support group.

Once these three groups are defined, the product group controls can be set to:

```
Product A:
AccessA: ENTRY, MANDATORY/MANDATORY
Product B:
AccessB: ENTRY, MANDATORY/MANDATORY
```

Perhaps the "Support Group" wants more control. For example, the "Support Group" could be permitted to make bugs inaccessible to users of both groups "AccessA" and "AccessB". Then, the "Support Group" could be permitted to publish bugs relevant to all users in a third product (let's call it "Product Common") that is read-only to anyone outside the "Support Group". In this way the "Support Group" could control bugs that should be seen by both groups. That configuration would be:

```
Product A:
AccessA: ENTRY, MANDATORY/MANDATORY
Support: SHOWN/NA
Product B:
AccessB: ENTRY, MANDATORY/MANDATORY
Support: SHOWN/NA
Product Common:
Support: ENTRY, DEFAULT/MANDATORY, CANEDIT
```

### Make a Product Read Only

Sometimes a product is retired and should no longer have new bugs filed against it (for example, an older version of a software product that is no longer supported). A product can be made read-only by creating a group called "readonly" and adding products to the group as needed:

```
Product A:
ReadOnly: ENTRY, NA/NA, CANEDIT
```

**Note:** For more information on Groups outside of how they relate to products see *Groups and Security*.

## 3.4.3 Components

Components are subsections of a Product. E.g. the computer game you are designing may have a "UI" component, an "API" component, a "Sound System" component, and a "Plugins" component, each overseen by a different programmer. It often makes sense to divide Components in Bugzilla according to the natural divisions of responsibility within your Product or company.

Each component has a default assignee and, if you turned it on in the *Parameters*, a QA Contact. The default assignee should be the primary person who fixes bugs in that component. The QA Contact should be the person who will ensure these bugs are completely fixed. The Assignee, QA Contact, and Reporter will get email when new bugs are created in this Component and when these bugs change. Default Assignee and Default QA Contact fields only dictate the *default assignments*; these can be changed on bug submission, or at any later point in a bug's life.

To create a new Component:

1. Select the `Edit components` link from the `Edit product` page.
2. Select the `Add` link in the bottom right.
3. Fill out the `Component` field, a short `Description`, the `Default Assignee`, `Default CC List`, and `Default QA Contact` (if enabled). The `Component Description` field may contain a limited subset of HTML tags. The `Default Assignee` field must be a login name already existing in the Bugzilla database.

## 3.4.4 Versions

Versions are the revisions of the product, such as "Flinders 3.1", "Flinders 95", and "Flinders 2000". Version is not a multi-select field; the usual practice is to select the earliest version known to have the bug.

To create and edit Versions:

1. From the "Edit product" screen, select "Edit Versions".
2. You will notice that the product already has the default version "undefined". Click the "Add" link in the bottom right.
3. Enter the name of the Version. This field takes text only. Then click the "Add" button.

### 3.4.5 Milestones

Milestones are “targets” that you plan to get a bug fixed by. For example, if you have a bug that you plan to fix for your 3.0 release, it would be assigned the milestone of 3.0.

---

**Note:** Milestone options will only appear for a Product if you turned on the usetargetmilestone parameter in the “Bug Fields” tab of the *Parameters* page.

---

To create new Milestones and set Default Milestones:

1. Select “Edit milestones” from the “Edit product” page.
2. Select “Add” in the bottom right corner.
3. Enter the name of the Milestone in the “Milestone” field. You can optionally set the “sortkey”, which is a positive or negative number (-32768 to 32767) that defines where in the list this particular milestone appears. This is because milestones often do not occur in alphanumeric order; for example, “Future” might be after “Release 1.2”. Select “Add”.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 3.5 Flags

If you have the editcomponents permission, you can edit Flag Types from the main administration page. Clicking the *Flags* link will bring you to the *Administer Flag Types* page. Here, you can select whether you want to create (or edit) a Bug flag or an Attachment flag.

The two flag types have the same administration interface, and the interface for creating a flag and editing a flag have the same set of fields.

### 3.5.1 Flag Properties

**Name**
This is the name of the flag. This will be displayed to Bugzilla users who are looking at or setting the flag. The name may contain any valid Unicode characters except commas and spaces.

**Description**
The description describes the flag in more detail. It is visible in a tooltip when hovering over a flag either in the *Show Bug* or *Edit Attachment* pages. This field can be as long as you like and can contain any character you want.

**Category**
You can set a flag to be visible or not visible on any combination of products and components.

Default behavior for a newly created flag is to appear on all products and all components, which is why `__Any__:__Any__` is already entered in the *Inclusions* box. If this is not your desired behavior, you must either set some exclusions (for products on which you don’t want the flag to appear), or you must remove `__Any__:__Any__` from the *Inclusions* box and define products/components specifically for this flag.

To create an Inclusion, select a Product from the top drop-down box. You may also select a specific component from the bottom drop-down box. (Setting `__Any__` for Product translates to “all the products in this Bugzilla”. Selecting `__Any__` in the Component field means “all components in the selected product.”) Selections made, press *Include*, and your Product/Component pairing will show up in the *Inclusions* box on the right.

To create an Exclusion, the process is the same: select a Product from the top drop-down box, select a specific component if you want one, and press *Exclude*. The Product/Component pairing will show up in the *Exclusions* box on the right.

This flag *will* appear and *can* be set for any products/components appearing in the *Inclusions* box (or which fall under the appropriate `__Any__`). This flag *will not* appear (and therefore *cannot* be set) on any products appearing in the *Exclusions* box. *IMPORTANT: Exclusions override inclusions.*

You may select a Product without selecting a specific Component, but you cannot select a Component without a Product. If you do so, Bugzilla will display an error message, even if all your products have a component by that name. You will also see an error if you select a Component that does not belong to the selected Product.

*Example:* Let's say you have a product called `Jet Plane` that has thousands of components. You want to be able to ask if a problem should be fixed in the next model of plane you release. We'll call the flag `fixInNext`. However, one component in `Jet Plane` is called `Pilot`, and it doesn't make sense to release a new pilot, so you don't want to have the flag show up in that component. So, you include `Jet Plane:__Any__` and you exclude `Jet Plane:Pilot`.

**Sort Key**
Flags normally show up in alphabetical order. If you want them to show up in a different order, you can use this key set the order on each flag. Flags with a lower sort key will appear before flags with a higher sort key. Flags that have the same sort key will be sorted alphabetically.

**Active**
Sometimes you might want to keep old flag information in the Bugzilla database but stop users from setting any new flags of this type. To do this, uncheck *active*. Deactivated flags will still show up in the UI if they are `?`, `+`, or `-`, but they may only be cleared (unset) and cannot be changed to a new value. Once a deactivated flag is cleared, it will completely disappear from a bug/attachment and cannot be set again.

**Requestable**
New flags are, by default, "requestable", meaning that they offer users the `?` option, as well as `+` and `-`. To remove the `?` option, uncheck "requestable".

**Specifically Requestable**
By default this box is checked for new flags, meaning that users may make flag requests of specific individuals. Unchecking this box will remove the text box next to a flag; if it is still requestable, then requests cannot target specific users and are open to anyone (called a request "to the wind" in Bugzilla). Removing this after specific requests have been made will not remove those requests; that data will stay in the database (though it will no longer appear to the user).

**Multiplicable**
Any flag with *Multiplicable:guilabel:* set (default for new flags is 'on') may be set more than once. After being set once, an unset flag of the same type will appear below it with "addl." (short for "additional") before the name. There is no limit to the number of times a Multiplicable flags may be set on the same bug/attachment.

**CC List**
If you want certain users to be notified every time this flag is set to `?`, `-`, or `+`, or is unset, add them here. This is a comma-separated list of email addresses that need not be restricted to Bugzilla usernames.

**Grant Group**
When this field is set to some given group, only users in the group can set the flag to `+` and `-`. This field does not affect who can request or cancel the flag. For that, see the *Request Group* field below. If this field is left blank, all users can set or delete this flag. This field is useful for restricting which users can approve or reject requests.

**Request Group**
When this field is set to some given group, only users in the group can request or cancel this flag. Note that this field has no effect if the *Grant Group* field is empty. You can set the value of this field to a different group, but both fields have to be set to a group for this field to have an effect.

### 3.5.2 Deleting a Flag

When you are at the *Administer Flag Types* screen, you will be presented with a list of Bug flags and a list of Attachment Flags.

To delete a flag, click on the *Delete* link next to the flag description.

**Warning:** Once you delete a flag, it is *gone* from your Bugzilla. All the data for that flag will be deleted. Everywhere that flag was set, it will disappear, and you cannot get that data back. If you want to keep flag data, but don't want anybody to set any new flags or change current flags, unset *active* in the flag Edit form.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 3.6 Custom Fields

Custom Fields are fields defined by the administrator, in addition to those which come with Bugzilla by default. Custom Fields are treated like any other field—they can be set in bugs and used for search queries.

Administrators should keep in mind that adding too many fields can make the user interface more complicated and harder to use. Custom Fields should be added only when necessary and with careful consideration.

**Note:** Before adding a Custom Field, make sure that Bugzilla cannot already do the desired behavior. Many Bugzilla options are not enabled by default, and many times Administrators find that simply enabling certain options that already exist is sufficient.

Administrators can manage Custom Fields using the `Custom Fields` link on the Administration page. The Custom Fields administration page displays a list of Custom Fields, if any exist, and a link to "Add a new custom field".

### 3.6.1 Adding Custom Fields

To add a new Custom Field, click the "Add a new custom field" link. This page displays several options for the new field, described below.

The following attributes must be set for each new custom field:

- *Name:* The name of the field in the database, used internally. This name MUST begin with `cf_` to prevent confusion with standard fields. If this string is omitted, it will be automatically added to the name entered.
- *Description:* A brief string used as the label for this Custom Field. That is the string that users will see, and it should be short and explicit.
- *Type:* The type of field to create. There are several types available:

  **Bug ID:**
  A field where you can enter the ID of another bug from the same Bugzilla installation. To point to a bug in a remote installation, use the See Also field instead.

  **Large Text Box:**
  A multiple line box for entering free text.

  **Free Text:**
  A single line box for entering free text.

**Multiple-Selection Box:**
A list box where multiple options can be selected. After creating this field, it must be edited to add the selection options. See *Viewing/Editing Legal Values* for information about editing legal values.

**Drop Down:**
A list box where only one option can be selected. After creating this field, it must be edited to add the selection options. See *Viewing/Editing Legal Values* for information about editing legal values.

**Date/Time:**
A date field. This field appears with a calendar widget for choosing the date.

- *Sortkey:* Integer that determines in which order Custom Fields are displayed in the User Interface, especially when viewing a bug. Fields with lower values are displayed first.
- *Reverse Relationship Description:* When the custom field is of type `Bug ID`, you can enter text here which will be used as label in the referenced bug to list bugs which point to it. This gives you the ability to have a mutual relationship between two bugs.
- *Can be set on bug creation:* Boolean that determines whether this field can be set on bug creation. If not selected, then a bug must be created before this field can be set. See *Filing a Bug* for information about filing bugs.
- *Displayed in bugmail for new bugs:* Boolean that determines whether the value set on this field should appear in bugmail when the bug is filed. This attribute has no effect if the field cannot be set on bug creation.
- *Is obsolete:* Boolean that determines whether this field should be displayed at all. Obsolete Custom Fields are hidden.
- *Is mandatory:* Boolean that determines whether this field must be set. For single and multi-select fields, this means that a (non-default) value must be selected; for text and date fields, some text must be entered.
- *Field only appears when:* A custom field can be made visible when some criteria is met. For instance, when the bug belongs to one or more products, or when the bug is of some given severity. If left empty, then the custom field will always be visible, in all bugs.
- *Field that controls the values that appear in this field:* When the custom field is of type `Drop Down` or `Multiple-Selection Box`, you can restrict the availability of the values of the custom field based on the value of another field. This criteria is independent of the criteria used in the `Field only appears when` setting. For instance, you may decide that some given value `valueY` is only available when the bug status is RESOLVED while the value `valueX` should always be listed. Once you have selected the field that should control the availability of the values of this custom field, you can edit values of this custom field to set the criteria; see *Viewing/Editing Legal Values*.

### 3.6.2 Editing Custom Fields

As soon as a Custom Field is created, its name and type cannot be changed. If this field is a drop-down menu, its legal values can be set as described in *Viewing/Editing Legal Values*. All other attributes can be edited as described above.

### 3.6.3 Deleting Custom Fields

Only custom fields that are marked as obsolete, and that have never been used, can be deleted completely (else the integrity of the bug history would be compromised). For custom fields marked as obsolete, a “Delete” link will appear in the `Action` column. If the custom field has been used in the past, the deletion will be rejected. Marking the field as obsolete, however, is sufficient to hide it from the user interface entirely.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 3.7 Field Values

Legal values for the operating system, platform, bug priority and severity, and custom fields of type `Drop Down` and `Multiple-Selection Box` (see *Custom Fields*), as well as the list of valid bug statuses and resolutions, can be customized from the same interface. You can add, edit, disable, and remove the values that can be used with these fields.

### 3.7.1 Viewing/Editing Legal Values

Editing legal values requires `admin` privileges. Select "Field Values" from the Administration page. A list of all fields, both system and Custom, for which legal values can be edited appears. Click a field name to edit its legal values.

There is no limit to how many values a field can have, but each value must be unique to that field. The sortkey is important to display these values in the desired order.

When the availability of the values of a custom field is controlled by another field, you can select from here which value of the other field must be set for the value of the custom field to appear.

### 3.7.2 Deleting Legal Values

Legal values from Custom Fields can be deleted, but only if the following two conditions are respected:

1. The value is not set as the default for the field.
2. No bug is currently using this value.

If any of these conditions is not respected, the value cannot be deleted. The only way to delete these values is to reassign bugs to another value and to set another value as default for the field.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 3.8 Workflow

The bug status workflow—which statuses are valid transitions from which other statuses—can be customized.

You need to begin by defining the statuses and resolutions you want to use (see *Field Values*). By convention, these are in all capital letters.

Only one bug status, UNCONFIRMED, can never be renamed nor deleted. However, it can be disabled entirely on a per-product basis (see *Classifications, Products, Components, Versions, and Milestones*). The status referred to by the duplicate_or_move_bug_status parameter, if set, is also undeletable. To make it deletable, simply set the value of that parameter to a different status.

Aside from the empty value, two resolutions, DUPLICATE and FIXED, cannot be renamed or deleted. (FIXED could be if we fixed bug 1007605.)

Once you have defined your statuses, you can configure the workflow of how a bug moves between them. The workflow configuration page displays all existing bug statuses twice: first on the left for the starting status, and on the top for the target status in the transition. If the checkbox is checked, then the transition from the left to the top status is legal; if it's unchecked, that transition is forbidden.

The status used as the duplicate_or_move_bug_status parameter (normally RESOLVED or its equivalent) is required to be a legal transition from every other bug status, and so this is enforced on the page.

The "View Comments Required on Status Transitions" link below the table lets you set which transitions require a comment from the user.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 3.9 Groups and Security

Groups allow for separating bugs into logical divisions. Groups are typically used to isolate bugs that should only be seen by certain people. For example, a company might create a different group for each one of its customers or partners. Group permissions could be set so that each partner or customer would only have access to their own bugs. Or, groups might be used to create variable access controls for different departments within an organization. Another common use of groups is to associate groups with products, creating isolation and access control on a per-product basis.

Groups and group behaviors are controlled in several places:

1. The group configuration page. To view or edit existing groups, or to create new groups, access the "Groups" link from the "Administration" page. This section of the manual deals primarily with the aspect of group controls accessed on this page.
2. Global configuration parameters. Bugzilla has several parameters that control the overall default group behavior and restriction levels. For more information on the parameters that control group behavior globally, see *Group Security*.
3. Product association with groups. Most of the functionality of groups and group security is controlled at the product level. Some aspects of group access controls for products are discussed in this section, but for more detail see *Assigning Group Controls to Products*.
4. Group access for users. See *Assigning Users to Groups* for details on how users are assigned group access.

Group permissions are such that if a bug belongs to a group, only members of that group can see the bug. If a bug is in more than one group, only members of *all* the groups that the bug is in can see the bug. For information on granting read-only access to certain people and full edit access to others, see *Assigning Group Controls to Products*.

---

**Note:** By default, bugs can also be seen by the Assignee, the Reporter, and everyone on the CC List, regardless of whether or not the bug would typically be viewable by them. Visibility to the Reporter and CC List can be overridden (on a per-bug basis) by bringing up the bug, finding the section that starts with `Users in the roles selected below...` and un-checking the box next to either 'Reporter' or 'CC List' (or both).

---

### 3.9.1 Creating Groups

To create a new group, follow the steps below:

1. Select the `Administration` link in the page footer, and then select the `Groups` link from the Administration page.
2. A table of all the existing groups is displayed. Below the table is a description of all the fields. To create a new group, select the `Add Group` link under the table of existing groups.
3. There are five fields to fill out. These fields are documented below the form. Choose a name and description for the group. Decide whether this group should be used for bugs (in all likelihood this should be selected). Optionally, choose a regular expression that will automatically add any matching users to the group, and choose an icon that will help identify user comments for the group. The regular expression can be useful, for example, to automatically put all users from the same company into one group (if the group is for a specific customer or partner).

**Note:** If `User RegExp` is filled out, users whose email addresses match the regular expression will automatically be members of the group as long as their email addresses continue to match the regular expression. If their email address changes and no longer matches the regular expression, they will be removed from the group. Versions 2.16 and older of Bugzilla did not automatically remove users whose email addresses no longer matched the RegExp.

**Warning:** If specifying a domain in the regular expression, end the regexp with a "$". Otherwise, when granting access to "@mycompany\.com", access will also be granted to 'badperson@mycompany.com.cracker.net'. Use the syntax, '@mycompany\.com$' for the regular expression.

4. After the new group is created, it can be edited for additional options. The "Edit Group" page allows for specifying other groups that should be included in this group and which groups should be permitted to add and delete users from this group. For more details, see *Editing Groups and Assigning Group Permissions*.

## 3.9.2 Editing Groups and Assigning Group Permissions

To access the "Edit Groups" page, select the `Administration` link in the page footer, and then select the `Groups` link from the Administration page. A table of all the existing groups is displayed. Click on a group name you wish to edit or control permissions for.

The "Edit Groups" page contains the same five fields present when creating a new group. Below that are two additional sections, "Group Permissions" and "Mass Remove". The "Mass Remove" option simply removes all users from the group who match the regular expression entered. The "Group Permissions" section requires further explanation.

The "Group Permissions" section on the "Edit Groups" page contains four sets of permissions that control the relationship of this group to other groups. If the usevisibilitygroups parameter is in use (see *Parameters*) two additional sets of permissions are displayed. Each set consists of two select boxes. On the left, a select box with a list of all existing groups. On the right, a select box listing all groups currently selected for this permission setting (this box will be empty for new groups). The way these controls allow groups to relate to one another is called *inheritance*. Each of the six permissions is described below.

***Groups That Are a Member of This Group***
Members of any groups selected here will automatically have membership in this group. In other words, members of any selected group will inherit membership in this group.

***Groups That This Group Is a Member Of***
Members of this group will inherit membership to any group selected here. For example, suppose the group being edited is an Admin group. If there are two products (Product1 and Product2) and each product has its own group (Group1 and Group2), and the Admin group should have access to both products, simply select both Group1 and Group2 here.

***Groups That Can Grant Membership in This Group***
The members of any group selected here will be able add users to this group, even if they themselves are not in this group.

***Groups That This Group Can Grant Membership In***
Members of this group can add users to any group selected here, even if they themselves are not in the selected groups.

***Groups That Can See This Group***
Members of any selected group can see the users in this group. This setting is only visible if the usevisibilitygroups parameter is enabled on the Bugzilla Configuration page. See *Parameters* for information on configuring Bugzilla.

***Groups That This Group Can See***

Members of this group can see members in any of the selected groups. This setting is only visible if the usevisibilitygroups parameter is enabled on the the Bugzilla Configuration page. See *Parameters* for information on configuring Bugzilla.

### 3.9.3 Assigning Users to Groups

A User can become a member of a group in several ways:

1. The user can be explicitly placed in the group by editing the user's profile. This can be done by accessing the "Users" page from the "Administration" page. Use the search form to find the user you want to edit group membership for, and click on their email address in the search results to edit their profile. The profile page lists all the groups and indicates if the user is a member of the group either directly or indirectly. More information on indirect group membership is below. For more details on User Administration, see *Users*.
2. The group can include another group of which the user is a member. This is indicated by square brackets around the checkbox next to the group name in the user's profile. See *Editing Groups and Assigning Group Permissions* for details on group inheritance.
3. The user's email address can match the regular expression that has been specified to automatically grant membership to the group. This is indicated by "*" around the check box by the group name in the user's profile. See *Creating Groups* for details on the regular expression option when creating groups.

### 3.9.4 Assigning Group Controls to Products

The primary functionality of groups is derived from the relationship of groups to products. The concepts around segregating access to bugs with product group controls can be confusing. For details and examples on this topic, see *Assigning Group Controls to Products*.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 3.10 Keywords

The administrator can define keywords which can be used to tag and categorize bugs. For example, the keyword "regression" is commonly used. A company might have a policy stating all regressions must be fixed by the next release—this keyword can make tracking those bugs much easier. Keywords are global, rather than per product.

Keywords can be created, edited, or deleted by clicking the "Keywords" link in the admin page. There are two fields for each keyword—the keyword itself and a brief description. Currently keywords cannot be marked obsolete to prevent future usage.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 3.11 Whining

Whining is a feature in Bugzilla that can regularly annoy users at specified times. Using this feature, users can execute saved searches at specific times (e.g. the 15th of the month at midnight) or at regular intervals (e.g. every 15 minutes on Sundays). The results of the searches are sent to the user, either as a single email or as one email per bug, along with some descriptive text.

**Warning:** Throughout this section it will be assumed that all users are members of the bz_canusewhines group, membership in which is required in order to use the Whining system. You can easily make all users members of the bz_canusewhines group by setting the User RegExp to ".*" (without the quotes).

Also worth noting is the bz_canusewhineatothers group. Members of this group can create whines for any user or group in Bugzilla using an extended form of the whining interface. Features only available to members of the bz_canusewhineatothers group will be noted in the appropriate places.

**Note:** For whining to work, a special Perl script must be executed at regular intervals. More information on this is available in installation-whining.

**Note:** This section does not cover the whineatnews.pl script. See installation-whining-cron for more information on The Whining Cron.

### 3.11.1 The Event

The whining system defines an "Event" as one or more queries being executed at regular intervals, with the results of said queries (if there are any) being emailed to the user. Events are created by clicking on the "Add new event" button.

Once a new event is created, the first thing to set is the "Email subject line". The contents of this field will be used in the subject line of every email generated by this event. In addition to setting a subject, space is provided to enter some descriptive text that will be included at the top of each message (to help you in understanding why you received the email in the first place).

The next step is to specify when the Event is to be run (the Schedule) and what searches are to be performed (the Searches).

### 3.11.2 Whining Schedule

Each whining event is associated with zero or more schedules. A schedule is used to specify when the search (specified below) is to be run. A new event starts out with no schedules (which means it will never run, as it is not scheduled to run). To add a schedule, press the "Add a new schedule" button.

Each schedule includes an interval, which you use to tell Bugzilla when the event should be run. An event can be run on certain days of the week, certain days of the month, during weekdays (defined as Monday through Friday), or every day.

**Warning:** Be careful if you set your event to run on the 29th, 30th, or 31st of the month, as your event may not run exactly when expected. If you want your event to run on the last day of the month, select "Last day of the month" as the interval.

Once you have specified the day(s) on which the event is to be run, you should now specify the time at which the event is to be run. You can have the event run at a certain hour on the specified day(s), or every hour, half-hour, or quarter-hour on the specified day(s).

If a single schedule does not execute an event as many times as you would want, you can create another schedule for the same event. For example, if you want to run an event on days whose numbers are divisible by seven, you would need to add four schedules to the event, setting the schedules to run on the 7th, 14th, 21st, and 28th (one day per schedule) at whatever time (or times) you choose.

**Note:** If you are a member of the bz_canusewhineatothers group, then you will be presented with another option: “Mail to”. Using this you can control who will receive the emails generated by this event. You can choose to send the emails to a single user (identified by email address) or a single group (identified by group name). To send to multiple users or groups, create a new schedule for each additional user/group.

### 3.11.3 Whining Searches

Each whining event is associated with zero or more searches. A search is any saved search to be run as part of the specified schedule (see above). You start out without any searches associated with the event (which means that the event will not run, as there will never be any results to return). To add a search, press the “Add a search” button.

The first field to examine in your newly added search is the Sort field. Searches are run, and results included, in the order specified by the Sort field. Searches with smaller Sort values will run before searches with bigger Sort values.

The next field to examine is the Search field. This is where you choose the actual search that is to be run. Instead of defining search parameters here, you are asked to choose from the list of saved searches (the same list that appears at the bottom of every Bugzilla page). You are only allowed to choose from searches that you have saved yourself (the default saved search, “My Bugs”, is not a valid choice). If you do not have any saved searches, you can take this opportunity to create one (see *Bug Lists*).

**Note:** When running searches, the whining system acts as if you are the user executing the search. This means that the whining system will ignore bugs that match your search but that you cannot access.

Once you have chosen the saved search to be executed, give the search a descriptive title. This title will appear in the email, above the results of the search. If you choose “One message per bug”, the search title will appear at the top of each email that contains a bug matching your search.

Finally, decide if the results of the search should be sent in a single email, or if each bug should appear in its own email.

**Warning:** Think carefully before checking the “One message per bug” box. If you create a search that matches thousands of bugs, you will receive thousands of emails!

### 3.11.4 Saving Your Changes

Once you have defined at least one schedule and created at least one search, go ahead and "Update/Commit". This will save your Event and make it available for immediate execution.

**Note:** If you ever feel like deleting your event, you may do so using the "Remove Event" button in the upper-right corner of each Event. You can also modify an existing event, so long as you "Update/Commit" after completing your modifications.

## 3.12 Quips

Quips are small user-defined messages (often quotes or witty sayings) that can be configured to appear at the top of search results. Each Bugzilla installation has its own specific quips. Whenever a quip needs to be displayed, a random selection is made from the pool of already existing quips.

Quip submission is controlled by quip_list_entry_control parameter. It has several possible values: open, moderated, or closed. In order to enable quips approval you need to set this parameter to "moderated". In this way, users are free to submit quips for addition, but an administrator must explicitly approve them before they are actually used.

In order to see the user interface for the quips, you can click on a quip when it is displayed together with the search results. You can also go directly to the quips.cgi URL (prefixed with the usual web location of the Bugzilla installation). Once the quip interface is displayed, the "view and edit the whole quip list" link takes you to the quips administration page, which lists all quips available in the database.

Next to each quip there is a checkbox, under the "Approved" column. Quips that have this checkbox checked are already approved and will appear next to the search results. The ones that have it unchecked are still preserved in the database but will not appear on search results pages. User submitted quips have initially the checkbox unchecked.

Also, there is a delete link next to each quip, which can be used in order to permanently delete a quip.

Display of quips is controlled by the *display_quips* user preference. Possible values are "on" and "off".

## 3.13 Installed Extensions

Bugzilla can be enhanced using extensions (see *Extensions*). If an extension comes with documentation in the appropriate format, and you build your own copy of the Bugzilla documentation using `makedocs.pl`, then the documentation for your installed extensions will show up here.

Your Bugzilla installation has the following extensions available (as of the last time you compiled the documentation):

CHAPTER

FOUR

# INTEGRATION AND CUSTOMIZATION GUIDE

You may find that Bugzilla already does what you want it to do, you just need to configure it correctly. Read the *Administration Guide* sections carefully to see if that's the case for you. If not, then this chapter explains how to use the available mechanisms for integration and customization.

## 4.1 Customization FAQ

How do I…

**…add a new field on a bug?**
Use *Custom Fields* or, if you just want new form fields on bug entry but don't need Bugzilla to track the field separately thereafter, you can use a *custom bug entry form*.

**…change the name of a built-in bug field?**
*Edit* the relevant value in the template `template/en/default/global/field-descs.none.tmpl`.

**…use a word other than 'bug' to describe bugs?**
*Edit or override* the appropriate values in the template `template/en/default/global/variables.none.tmpl`.

**…call the system something other than 'Bugzilla'?**
*Edit or override* the appropriate value in the template `template/en/default/global/variables.none.tmpl`.

**…alter who can change what field when?**
See *Altering Who Can Change What*.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 4.2 Languages

Bugzilla's templates can be localized, although it's a big job. If you have a localized set of templates for your version of Bugzilla, Bugzilla can support multiple languages at once. In that case, Bugzilla honours the user's `Accept-Language` HTTP header when deciding which language to serve. If multiple languages are installed, a menu will display in the header allowing the user to manually select a different language. If they do this, their choice will override the `Accept-Language` header.

Many language templates can be obtained from the localization section of the Bugzilla website. Instructions for submitting new languages are also available from that location. There's also a list of localization teams; you might want to contact someone to ask about the status of their localization.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 4.3 Skins

Bugzilla supports skins - ways of changing the look of the UI without altering its underlying structure. It ships with two - “Classic” and “Dusk”. You can find some more listed on the wiki, and there are a couple more which are part of bugzilla.mozilla.org. However, in each case you may need to check that the skin supports the version of Bugzilla you have.

To create a new custom skin, make a directory that contains all the same CSS file names as `skins/standard/`, and put your directory in `skins/contrib/`. Then, add your CSS to the appropriate files.

After you put the directory there, make sure to run `checksetup.pl` so that it can set the file permissions correctly.

After you have installed the new skin, it will show up as an option in the user’s *Preferences*, on the *General* tab. If you would like to force a particular skin on all users, just select that skin in the *Default Preferences* in the *Administration* UI, and then uncheck “Enabled” on the preference, so users cannot change it.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 4.4 Templates

Bugzilla uses a system of templates to define its user interface. The standard templates can be modified, replaced or overridden. You can also use template hooks in an *extension* to add or modify the behavior of templates using a stable interface.

### 4.4.1 Template Directory Structure

The template directory structure starts with top level directory named `template`, which contains a directory for each installed localization. Bugzilla comes with English templates, so the directory name is `en`, and we will discuss `template/en` throughout the documentation. Below `template/en` is the `default` directory, which contains all the standard templates shipped with Bugzilla.

**Warning:** A directory `data/template` also exists; this is where Template Toolkit puts the compiled versions (i.e. Perl code) of the templates. *Do not* directly edit the files in this directory, or all your changes will be lost the next time Template Toolkit recompiles the templates.

### 4.4.2 Choosing a Customization Method

If you want to edit Bugzilla’s templates, the first decision you must make is how you want to go about doing so. There are three choices, and which you use depends mainly on the scope of your modifications, and the method you plan to use to upgrade Bugzilla.

1. You can directly edit the templates found in `template/en/default`.
2. You can copy the templates to be modified into a mirrored directory structure under `template/en/custom`. Templates in this directory structure automatically override any identically-named and identically-located templates in the `template/en/default` directory. (The `custom` directory does not exist by default and must be created if you want to use it.)

3. You can use the hooks built into many of the templates to add or modify the UI from an *extension*. Hooks generally don't go away and have a stable interface.

The third method is the best if there are hooks in the appropriate places and the change you want to do is possible using hooks. It's not very easy to modify existing UI using hooks; they are most commonly used for additions. You can make modifications if you add JS code which then makes the modifications when the page is loaded. You can remove UI by adding CSS to hide it.

Unlike code hooks, there is no requirement to document template hooks, so you just have to open up the template and see (search for `Hook.process`).

If there are no hooks available, then the second method of customization should be used if you are going to make major changes, because it is guaranteed that the contents of the `custom` directory will not be touched during an upgrade, and you can then decide whether to revert to the standard templates, continue using yours, or make the effort to merge your changes into the new versions by hand. It's also good for entirely new files, and for a few files like `bug/create/user-message.html.tmpl` which are designed to be entirely replaced.

Using the second method, your user interface may break if incompatible changes are made to the template interface. Templates do change regularly and so interface changes are not individually documented, and you would need to work out what had changed and adapt your template accordingly.

For minor changes, the convenience of the first method is hard to beat. When you upgrade Bugzilla, **`git`** will merge your changes into the new version for you. On the downside, if the merge fails then Bugzilla will not work properly until you have fixed the problem and re-integrated your code.

Also, you can see what you've changed using **`git diff`**, which you can't if you fork the file into the `custom` directory.

### 4.4.3 How To Edit Templates

**Note:** If you are making template changes that you intend on submitting back for inclusion in standard Bugzilla, you should read the relevant sections of the Developers' Guide.

Bugzilla uses a templating system called Template Toolkit. The syntax of the language is beyond the scope of this guide. It's reasonably easy to pick up by looking at the current templates; or, you can read the manual, available on the Template Toolkit home page.

One thing you should take particular care about is the need to properly HTML filter data that has been passed into the template. This means that if the data can possibly contain special HTML characters such as <, and the data was not intended to be HTML, they need to be converted to entity form, i.e. `&lt;`. You use the `html` filter in the Template Toolkit to do this (or the `uri` filter to encode special characters in URLs). If you forget, you may open up your installation to cross-site scripting attacks.

You should run **`./checksetup.pl`** after editing any templates. Failure to do so may mean either that your changes are not picked up, or that the permissions on the edited files are wrong so the webserver can't read them.

### 4.4.4 Template Formats and Types

Some CGI's have the ability to use more than one template. For example, `buglist.cgi` can output itself as two formats of HTML (complex and simple). Each of these is a separate template. The mechanism that provides this feature is extensible - you can create new templates to add new formats.

You might use this feature to e.g. add a custom bug entry form for a particular subset of users or a particular type of bug.

Bugzilla can also support different types of output - e.g. bugs are available as HTML and as XML, and this mechanism is extensible also to add new content types. However, instead of using such interfaces or enhancing Bugzilla to add more, you would be better off using the *WebService API Reference* to integrate with Bugzilla.

To see if a CGI supports multiple output formats and types, grep the CGI for `get_format`. If it's not present, adding multiple format/type support isn't too hard - see how it's done in other CGIs, e.g. `config.cgi`.

To make a new format template for a CGI which supports this, open a current template for that CGI and take note of the INTERFACE comment (if present.) This comment defines what variables are passed into this template. If there isn't one, I'm afraid you'll have to read the template and the code to find out what information you get.

Write your template in whatever markup or text style is appropriate.

You now need to decide what content type you want your template served as. The content types are defined in the `Bugzilla/Constants.pm` file in the `contenttypes` constant. If your content type is not there, add it. Remember the three- or four-letter tag assigned to your content type. This tag will be part of the template filename.

Save your new template as `<stubname>-<formatname>.<contenttypetag>.tmpl`. Try out the template by calling the CGI as `<cginame>.cgi?format=<formatname>`. Add `&ctype=<type>` if the type is not HTML.

### 4.4.5 Particular Templates

There are a few templates you may be particularly interested in customizing for your installation.

**`index.html.tmpl`:**
: This is the Bugzilla front page.

**`global/header.html.tmpl`:**
: This defines the header that goes on all Bugzilla pages. The header includes the banner, which is what appears to users and is probably what you want to edit instead. However the header also includes the HTML HEAD section, so you could for example add a stylesheet or META tag by editing the header.

**`global/banner.html.tmpl`:**
: This contains the `banner`, the part of the header that appears at the top of all Bugzilla pages. The default banner is reasonably barren, so you'll probably want to customize this to give your installation a distinctive look and feel. It is recommended you preserve the Bugzilla version number in some form so the version you are running can be determined, and users know what docs to read.

**`global/footer.html.tmpl`:**
: This defines the footer that goes on all Bugzilla pages. Editing this is another way to quickly get a distinctive look and feel for your Bugzilla installation.

**`global/variables.none.tmpl`:**
: This allows you to change the word 'bug' to something else (e.g. "issue") throughout the interface, and also to change the name Bugzilla to something else (e.g. "FooCorp Bug Tracker").

**`list/table.html.tmpl`:**
: This template controls the appearance of the bug lists created by Bugzilla. Editing this template allows per-column control of the width and title of a column, the maximum display length of each entry, and the wrap behavior of long entries. For long bug lists, Bugzilla inserts a 'break' every 100 bugs by default; this behavior is also controlled by this template, and that value can be modified here.

**bug/create/user-message.html.tmpl:**
This is a message that appears near the top of the bug reporting page. By modifying this, you can tell your users how they should report bugs.

**bug/process/midair.html.tmpl:**
This is the page used if two people submit simultaneous changes to the same bug. The second person to submit their changes will get this page to tell them what the first person did, and ask if they wish to overwrite those changes or go back and revisit the bug. The default title and header on this page read "Mid-air collision detected!" If you work in the aviation industry, or other environment where this might be found offensive (yes, we have true stories of this happening) you'll want to change this to something more appropriate for your environment.

**bug/create/create.html.tmpl and bug/create/comment.txt.tmpl:**
You may not wish to go to the effort of creating custom fields in Bugzilla, yet you want to make sure that each bug report contains a number of pieces of important information for which there is not a special field. The bug entry system has been designed in an extensible fashion to enable you to add arbitrary HTML widgets, such as drop-down lists or textboxes, to the bug entry page and have their values appear formatted in the initial comment.

An example of this is the guided bug submission form. The code for this comes with the Bugzilla distribution as an example for you to copy. It can be found in the files `create-guided.html.tmpl` and `comment-guided.html.tmpl`.

A hidden field that indicates the format should be added inside the form in order to make the template functional. Its value should be the suffix of the template filename. For example, if the file is called `create-guided.html.tmpl`, then

```
<input type="hidden" name="format" value="guided">
```

is used inside the form.

So to use this feature, create a custom template for `enter_bug.cgi`. The default template, on which you could base it, is `default/bug/create/create.html.tmpl`. Call it `custom/bug/create/create-<formatname>.html.tmpl`, and in it, add form inputs for each piece of information you'd like collected - such as a build number, or set of steps to reproduce.

Then, create a template based on `default/bug/create/comment.txt.tmpl`, and call it `custom/bug/create/comment-<formatname>.txt.tmpl`. It needs a couple of lines of boilerplate at the top like this:

```
[% USE Bugzilla %]
[% cgi = Bugzilla.cgi %
```

Then, this template can reference the form fields you have created using the syntax `[% cgi.param("field_name") %]`. When a bug report is submitted, the initial comment attached to the bug report will be formatted according to the layout of this template.

For example, if your custom enter_bug template had a field:

```
<input type="text" name="buildid" size="30">
```

and then your comment.txt.tmpl had:

```
[% USE Bugzilla %]
[% cgi = Bugzilla.cgi %]
Build Identifier: [%+ cgi.param("buildid") %]
```

then something like:

```
Build Identifier: 20140303
```

would appear in the initial comment.

This system allows you to gather structured data in bug reports without the overhead and UI complexity of a large number of custom fields.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 4.5 Extensions

One of the best ways to customize Bugzilla is by using a Bugzilla Extension. Extensions can modify both the code and UI of Bugzilla in a way that can be distributed to other Bugzilla users and ported forward to future versions of Bugzilla with minimal effort. We maintain a list of available extensions written by other people on our wiki. You would need to make sure that the extension in question works with your version of Bugzilla.

Or, you can write your own extension. See the Bugzilla Extension documentation for the core documentation on how to do that. It would make sense to read the section on *Templates*. There is also a sample extension in `$BUGZILLA_HOME/extensions/Example/` which gives examples of how to use all the code hooks.

This section explains how to achieve some common tasks using the Extension APIs.

### 4.5.1 Adding A New Page to Bugzilla

There are occasions where it's useful to add a new page to Bugzilla which has little or no relation to other pages, and perhaps doesn't use very much Bugzilla data. A help page, or a custom report for example. The best mechanism for this is to use `page.cgi` and the `page_before_template` hook.

### 4.5.2 Altering Data On An Existing Page

The `template_before_process` hook can be used to tweak the data displayed on a particular existing page, if you know what template is used. It has access to all the template variables before they are passed to the templating engine.

### 4.5.3 Adding New Fields To Bugs

To add new fields to a bug, you need to do the following:

- Add an `install_update_db` hook to add the fields by calling `Bugzilla::Field->create` (only if the field doesn't already exist). Here's what it might look like for a single field:

```
my $field = new Bugzilla::Field({ name => $name });
return if $field;

$field = Bugzilla::Field->create({
    name        => $name,
    description => $description,
    type        => $type,        # From list in Constants.pm
    enter_bug   => 0,
    buglist     => 0,
    custom      => 1,
});
```

- Push the name of the field onto the relevant arrays in the `bug_columns` and `bug_fields` hooks.

- If you want direct accessors, or other functions on the object, you need to add a BEGIN block to your Extension.pm:

```
BEGIN {
   *Bugzilla::Bug::is_foopy = \&_bug_is_foopy;
}

...

sub _bug_is_foopy {
    return $_[0]->{'is_foopy'};
}
```

- You don't have to change `Bugzilla/DB/Schema.pm`.
- You can use `bug_end_of_create`, `bug_end_of_create_validators`, and `bug_end_of_update` to create or update the values for your new field.

### 4.5.4 Adding New Fields To Other Things

If you are adding the new fields to an object other than a bug, you need to go a bit lower-level. With reference to the instructions above:

- In `install_update_db`, use `bz_add_column` instead
- Push on the columns in `object_columns` and `object_update_columns` instead of `bug_columns`.
- Add validators for the values in `object_validators`

The process for adding accessor functions is the same.

You can use the hooks `object_end_of_create`, `object_end_of_create_validators`, `object_end_of_set_all`, and `object_end_of_update` to create or update the values for the new object fields you have added. In the hooks you can check the object type being operated on and skip any objects you don't care about. For example, if you added a new field to the `products` table:

```
sub object_end_of_create {
    my ($self, $args) = @_;
    my $class = $args->{'class'};
    my $object = $args->{'object'};
    if ($class->isa('Bugzilla::Product') {
        [...]
    }
}
```

You will need to do this filtering for most of the hooks whose names begin with `object_`.

### 4.5.5 Adding Admin Configuration Panels

If you add new functionality to Bugzilla, it may well have configurable options or parameters. The way to allow an administrator to set those is to add a new configuration panel.

As well as using the `config_add_panels` hook, you will need a template to define the UI strings for the panel. See the templates in `template/en/default/admin/params` for examples, and put your own template in `template/en/default/admin/params` in your extension's directory.

You can access param values from Templates using:

```
[% Param('param_name') %]
```

and from code using:

```
Bugzilla->params->{'param_name'}
```

### 4.5.6 Adding User Preferences

To add a new user preference:

- Call `add_setting('setting_name', ['some_option', 'another_option'], 'some_option')` in the `install_before_final_checks` hook. (The last parameter is the name of the option which should be the default.)
- Add descriptions for the identifiers for your setting and choices (setting_name, some_option etc.) to the hash defined in `global/setting-descs.none.tmpl`. Do this in a template hook: `hook/global/setting-descs-settings.none.tmpl`. Your code can see the hash variable; just set more members in it.
- To change behavior based on the setting, reference it in templates using `[% user.settings.setting_name.value %]`. Reference it in code using `$user->settings->{'setting_name'}->{'value'}`. The value will be one of the option tag names (e.g. some_option).

### 4.5.7 Altering Who Can Change What

Companies often have rules about which employees, or classes of employees, are allowed to change certain things in the bug system. For example, only the bug's designated QA Contact may be allowed to VERIFY the bug. Bugzilla has been designed to make it easy for you to write your own custom rules to define who is allowed to make what sorts of value transition.

By default, assignees, QA owners and users with *editbugs* privileges can edit all fields of bugs, except group restrictions (unless they are members of the groups they are trying to change). Bug reporters also have the ability to edit some fields, but in a more restrictive manner. Other users, without *editbugs* privileges, cannot edit bugs, except to comment and add themselves to the CC list.

Because this kind of change is such a common request, we have added a specific hook for it that *Extensions* can call. It's called `bug_check_can_change_field`, and it's documented in the Hooks documentation.

### 4.5.8 Checking Syntax

It's not immediately obvious how to check the syntax of your extension's Perl modules, if it contains any. Running **checksetup.pl** might do some of it, but the errors aren't necessarily massively informative.

```
perl -Mlib=lib -MBugzilla -e 'BEGIN { Bugzilla->extensions; } use
Bugzilla::Extension::ExtensionName::Class;'
```

(run from `$BUGZILLA_HOME`) is what you need.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 4.6 APIs

Bugzilla has a number of APIs that you can call in your code to extract information from and put information into Bugzilla. Some are deprecated and will soon be removed. Which one to use? Short answer: the *REST WebService API v1* should be used for all new integrations, but keep an eye out for version 2, coming soon.

The APIs currently available are as follows:

### 4.6.1 Ad-Hoc APIs

Various pages on Bugzilla are available in machine-parsable formats as well as HTML. For example, bugs can be downloaded as XML, and buglists as CSV. CSV is useful for spreadsheet import. There should be links on the HTML page to alternate data formats where they are available.

### 4.6.2 REST

Bugzilla has a *REST API* which is the currently-recommended API for integrating with Bugzilla. The current REST API is version 1. It is stable, and so will not be changed in a backwardly-incompatible way.

**This is the currently-recommended API for new development.**

Endpoint: `/rest`

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 4.7 Authentication Delegation via API Keys

Bugzilla provides a mechanism for web apps to request (with the user's consent) an API key. API keys allow the web app to perform any action as the user and are as a result very powerful. Because of this power, this feature is disabled by default.

### 4.7.1 Authentication Flow

The authentication process begins by directing the user to th the Bugzilla site's auth.cgi. For the sake of this example, our application's URL is *http://app.example.org* and the Bugzilla site is *http://bugzilla.mozilla.org*.

1. Provide a link or redirect the user to *https://bugzilla.mozilla.org/auth.cgi?callback=http://app.example.org/callback&description=*
2. Assuming the user is agreeable, the following will happen:

1. Bugzilla will issue a POST request to *http://app.example.org/callback* with a the request body data being a JSON object with keys *client_api_key* and *client_api_login*.
2. The callback, when responding to the POST request must return a JSON object with a key *result*. This result is intended to be a unique token used to identify this transaction.
3. Bugzilla will then cause the user agent to redirect (using a GET request) to *http://app.example.org/callback* with additional query string parameters *client_api_login* and *callback_result*.
4. At this point, the consumer now has the api key and login information. Be sure to compare the *callback_result* to whatever result was initially sent back to Bugzilla.

3. Finally, you should check that the API key and login are valid, using the *Who Am I* REST resource.

Your application should take measures to ensure when receiving a user at your callback URL that you previously redirected them to Bugzilla. The simplest method would be ensuring the callback URL always has the hostname and path you specified, with only the query string parameters varying.

The description should include the name of your application, in a form that will be recognizable to users. This description is used in the *API Keys tab* in the Preferences page.

The API key passed to the callback will be valid until the user revokes it.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 4.8 Adding an Auth0 Custom Social Integration

Bugzilla can be added as a 'Custom Social Connection'.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

CHAPTER

FIVE

# WEBSERVICE API REFERENCE

This Bugzilla installation has the following WebService APIs available (as of the last time you compiled the documentation):

## 5.1 Core API v1

### 5.1.1 General

This is the standard REST API for external programs that want to interact with Bugzilla. It provides a REST interface to various Bugzilla functions.

#### Basic Information

**Data Format**

The REST API only supports JSON input, and either JSON or JSONP output. So objects sent and received must be in JSON format.

If you need JSONP output, you must set the `Accept: application/javascript` HTTP header and add a `callback` parameter to name your callback.

Parameters may also be passed in as part of the query string for non-GET requests and will override any matching parameters in the request body.

Example request which returns the current version of Bugzilla:

```
GET /rest/version HTTP/1.1
Host: bugzilla.example.com
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "version" : "4.2.9+"
}
```

**Errors**

When an error occurs over REST, an object is returned with the key `error` set to `true`.

The error contents look similar to:

```
{
  "error": true,
  "message": "Some message here",
  "code": 123
}
```

To protect the application from large requests, Bugzilla returns a 302 redirect to the homepage when your query string is too long. The current limit is 10 KB, which can accept roughly 1,000 bug IDs in the `id` parameter for the `/rest/bug` method, but it could be smaller or may lead to a 414 URI Too Long HTTP error depending on the server configuration. Split your query into multiple requests if you encounter the issue.

### Common Data Types

The Bugzilla API uses the following various types of parameters:

| type | description |
|---|---|
| int | Integer. |
| double | A floating-point number. |
| string | A string. |
| email | A string representing an email address. This value, when returned, may be filtered based on if the user is logged in or not. |
| date | A specific date. Example format: `YYYY-MM-DD`. |
| datetime | A date/time. Timezone should be in UTC unless otherwise noted. Example format: `YYYY-MM-DDTHH24:MI:SSZ`. |
| boolean | `true` or `false`. |
| base64 | A base64-encoded string. This is the only way to transfer binary data via the API. |
| array | An array. There may be mixed types in an array. `[` and `]` are used to represent the beginning and end of arrays. |
| object | A mapping of keys to values. Called a "hash", "dict", or "map" in some other programming languages. The keys are strings, and the values can be any type. `{` and `}` are used to represent the beginning and end of objects. |

Parameters that are required will be displayed in **bold** in the parameters table for each API method.

### Authentication

Some methods do not require you to log in. An example of this is *Get Bug*. However, authenticating yourself allows you to see non-public information, for example, a bug that is not publicly visible.

To authenticate yourself, you will need to use API keys:

**API Keys**

You can specify 'X-BUGZILLA-API-KEY' header with the API key as a value to any request, and you will be authenticated as that user if the key is correct and has not been revoked.

You can set up an API key by using the *API Keys tab* in the Preferences pages.

API keys may also be requested via *Authentication Delegation*.

**WARNING**: It should be noted that additional authentication methods exist, but they are **not recommended** for use and are likely to be deprecated in future versions of BMO, due to security concerns. These additional methods include the following:

- username and password via `Bugzilla_login` and `Bugzilla_password` or simply `login` and `password` respectively in query parameters.
- username and password via `X-BUGZILLA-LOGIN` and `X-BUGZILLA-PASSWORD` headers respectively.
- api key via `Bugzilla_api_key` or simply `api_key` in query parameters.

### Useful Parameters

Many calls take common arguments. These are documented below and linked from the individual calls where these parameters are used.

**Including Fields**

Many calls return an array of objects with various fields in the objects. (For example, *Get Bug* returns a list of `bugs` that have fields like `id`, `summary`, `creation_time`, etc.)

These parameters allow you to limit what fields are present in the objects, to improve performance or save some bandwidth.

`include_fields`: The (case-sensitive) names of fields in the response data. Only the fields specified in the object will be returned, the rest will not be included. Fields should be comma delimited.

Invalid field names are ignored.

Example request to *Get User*:

```
GET /rest/user/1?include_fields=id,name
```

would return something like:

```
{
  "users" : [
    {
      "id" : 1,
      "name" : "user@domain.com"
    }
  ]
}
```

**Excluding Fields**

`exclude_fields`: The (case-sensitive) names of fields in the return value. The fields specified will not be included in the returned objects. Fields should be comma delimited.

Invalid field names are ignored.

Specifying fields here overrides `include_fields`, so if you specify a field in both, it will be excluded, not included.

Example request to *Get User*:

```
GET /rest/user/1?exclude_fields=name
```

would return something like:

```
{
  "users" : [
    {
      "id" : 1,
      "real_name" : "John Smith"
    }
  ]
}
```

Some calls support specifying "subfields". If a call states that it supports "subfield" restrictions, you can restrict what information is returned within the first field. For example, if you call *Get Product* with an `include_fields` of `components.name`, then only the component name would be returned (and nothing else). You can include the main field, and exclude a subfield.

There are several shortcut identifiers to ask for only certain groups of fields to be returned or excluded:

| value | description |
|---|---|
| _all | All possible fields are returned if this is specified in `include_fields`. |
| _default | Default fields are returned if `include_fields` is empty or this is specified. This is useful if you want the default fields in addition to a field that is not normally returned. |
| _extra | Extra fields are not returned by default and need to be manually specified in `include_fields` either by exact field name, or adding `_extra`. |
| _custom | Custom fields are normally returned by default unless this is added to `exclude_fields`. Also you can use it in `include_fields` if for example you want specific field names plus all custom fields. Custom fields are normally only relevant to bug objects. |

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 5.1.2 Attachments

The Bugzilla API for creating, changing, and getting the details of attachments.

### Get Attachment

This allows you to get data about attachments, given a list of bugs and/or attachment IDs. Private attachments will only be returned if you are in the appropriate group or if you are the submitter of the attachment.

**Request**

To get all current attachments for a bug:

```
GET /rest/bug/(bug_id)/attachment
```

To get a specific attachment based on attachment ID:

```
GET /rest/bug/attachment/(attachment_id)
```

One of the below must be specified.

| name | type | description |
|---|---|---|
| **bug_id** | int | Integer bug ID. |
| **attachment_id** | int | Integer attachment ID. |

**Response**

```
{
   "bugs" : {
      "1345" : [
         { (attachment) },
         { (attachment) }
      ],
      "9874" : [
         { (attachment) },
         { (attachment) }
      ],
   },
   "attachments" : {
      "234" : { (attachment) },
      "123" : { (attachment) },
   }
}
```

An object containing two elements: `bugs` and `attachments`.

The attachments for the bug that you specified in the `bug_id` argument in input are returned in `bugs` on output. `bugs` is a object that has integer bug IDs for keys and the values are arrays of objects as attachments. (Fields for attachments are described below.)

For the attachment that you specified directly in `attachment_id`, they are returned in `attachments` on output. This is a object where the attachment ids point directly to objects describing the individual attachment.

The fields for each attachment (where it says `(attachment)` in the sample response above) are:

| name | type | description |
|---|---|---|
| data | base64 | The raw data of the attachment, encoded as Base64. |
| size | int | The length (in bytes) of the attachment. |
| creation_time | datetime | The time the attachment was created. |
| last_change_time | datetime | The last time the attachment was modified. |
| id | int | The numeric ID of the attachment. |
| bug_id | int | The numeric ID of the bug that the attachment is attached to. |
| file_name | string | The file name of the attachment. |
| summary | string | A short string describing the attachment. |
| content_type | string | The MIME type of the attachment. |
| is_private | boolean | `true` if the attachment is private (only visible to a certain group called the “insidergroup”, `false` otherwise. |
| is_obsolete | boolean | `true` if the attachment is obsolete, `false` otherwise. |
| is_patch | boolean | `true` if the attachment is a patch, `false` otherwise. |
| creator | string | The login name of the user that created the attachment. |
| creator_detail | object | An object containing detailed user information for the creator. To see the keys included in the user detail object, see *Get Bug*. |
| flags | array | Array of objects, each containing the information about the flag currently set for each attachment. Each flag object contains items described in the Flag object below. |

Flag object:

| name | type | description |
|---|---|---|
| id | int | The ID of the flag. |
| name | string | The name of the flag. |
| type_id | int | The type ID of the flag. |
| creation_date | datetime | The timestamp when this flag was originally created. |
| modification_date | datetime | The timestamp when the flag was last modified. |
| status | string | The current status of the flag such as ?, +, or -. |
| setter | string | The login name of the user who created or last modified the flag. |
| requestee | string | The login name of the user this flag has been requested to be granted or denied. Note, this field is only returned if a requestee is set. |

**Errors**

This method can throw all the same errors as *Get Bug*. In addition, it can also throw the following error:

- 304 (Auth Failure, Attachment is Private) You specified the id of a private attachment in the "attachment_ids" argument, and you are not in the "insider group" that can see private attachments.

## Create Attachment

This allows you to add an attachment to a bug in Bugzilla.

**Request**

To create attachment on a current bug:

```
POST /rest/bug/(bug_id)/attachment
```

```
{
  "ids" : [ 35 ],
  "is_patch" : true,
  "comment" : "This is a new attachment comment",
  "summary" : "Test Attachment",
  "content_type" : "text/plain",
  "data" : "(Some base64 encoded content)",
  "file_name" : "test_attachment.patch",
  "obsoletes" : [],
  "is_private" : false,
  "flags" : [
    {
      "name" : "review",
      "status" : "?",
      "requestee" : "user@bugzilla.org",
      "new" : true
    }
  ]
}
```

The params to include in the POST body, as well as the returned data format, are the same as below. The `bug_id` param will be overridden as it it pulled from the URL path.

| name | type | description |
|---|---|---|
| **ids** | array | The IDs or aliases of bugs that you want to add this attachment to. The same attachment and comment will be added to all these bugs. |
| **data** | base64 | The content of the attachment. You must encode it in base64 using an appropriate client library such as `MIME::Base64` for Perl. |
| **file_name** | string | The "file name" that will be displayed in the UI for this attachment and also downloaded copies will be given. |
| **summary** | string | A short string describing the attachment. |
| **content_type** | string | The MIME type of the attachment, like `text/plain` or `image/png`. |
| comment | string | A comment to add along with this attachment. |
| is_patch | boolean | `true` if Bugzilla should treat this attachment as a patch. If you specify this, you do not need to specify a `content_type`. The `content_type` of the attachment will be forced to `text/plain`. Defaults to `false` if not specified. |
| is_private | boolean | `true` if the attachment should be private (restricted to the "insidergroup"), `false` if the attachment should be public. Defaults to `false` if not specified. |
| flags | array | Flags objects to add to the attachment. The object format is described in the Flag object below. |
| bug_flags | array | Flag objects to add to the attachment's bug. See the `flags` param for *Create Bug* for the object format. |

Flag object:

To create a flag, at least the `status` and the `type_id` or `name` must be provided. An optional requestee can be passed if the flag type is requestable to a specific user.

| name | type | description |
|---|---|---|
| name | string | The name of the flag type. |
| type_id | int | The internal flag type ID. |
| status | string | The flags new status (i.e. "?", "+", "-" or "X" to clear a flag). |
| requestee | string | The login of the requestee if the flag type is requestable to a specific user. |

**Response**

```
{
  "ids" : [
    "2797"
  ]
}
```

| name | type | description |
|---|---|---|
| ids | array | Attachment IDs created. |

**Errors**

This method can throw all the same errors as *Get Bug*, plus:

- 129 (Flag Status Invalid) The flag status is invalid.
- 130 (Flag Modification Denied) You tried to request, grant, or deny a flag but only a user with the required permissions may make the change.

- 131 (Flag not Requestable from Specific Person) You can't ask a specific person for the flag.
- 133 (Flag Type not Unique) The flag type specified matches several flag types. You must specify the type id value to update or add a flag.
- 134 (Inactive Flag Type) The flag type is inactive and cannot be used to create new flags.
- 140 (Markdown Disabled) You tried to set the "is_markdown" flag of the comment to true but the Markdown feature is not enabled.
- 600 (Attachment Too Large) You tried to attach a file that was larger than Bugzilla will accept.
- 601 (Invalid MIME Type) You specified a "content_type" argument that was blank, not a valid MIME type, or not a MIME type that Bugzilla accepts for attachments.
- 603 (File Name Not Specified) You did not specify a valid for the "file_name" argument.
- 604 (Summary Required) You did not specify a value for the "summary" argument.
- 606 (Empty Data) You set the "data" field to an empty string.

### Update Attachment

This allows you to update attachment metadata in Bugzilla.

**Request**

To update attachment metadata on a current attachment:

```
PUT /rest/bug/attachment/(attachment_id)
```

```
{
  "ids" : [ 2796 ],
  "summary" : "Test XML file",
  "comment" : "Changed this from a patch to a XML file",
  "content_type" : "text/xml",
  "is_patch" : 0
}
```

| name | type | description |
|---|---|---|
| **attachment_id** | int | Integer attachment ID. |
| **ids** | array | The IDs of the attachments you want to update. |

| name | type | description |
|---|---|---|
| file_name | string | The "file name" that will be displayed in the UI for this attachment. |
| summary | string | A short string describing the attachment. |
| comment | string | An optional comment to add to the attachment's bug. |
| content_type | string | The MIME type of the attachment, like `text/plain` or `image/png`. |
| is_patch | boolean | `true` if Bugzilla should treat this attachment as a patch. If you specify this, you do not need to specify a `content_type`. The `content_type` of the attachment will be forced to `text/plain`. |
| is_private | boolean | `true` if the attachment should be private (restricted to the "insidergroup"), `false` if the attachment should be public. |
| is_obsolete | boolean | `true` if the attachment is obsolete, `false` otherwise. |
| flags | array | An array of Flag objects with changes to the flags. The object format is described in the Flag object below. |
| bug_flags | array | An optional array of Flag objects with changes to the flags of the attachment's bug. See the `flags` param for *Update Bug* for the object format. |

Flag object:

The following values can be specified. At least the `status` and one of `type_id`, `id`, or `name` must be specified. If a type_id or name matches a single currently set flag, the flag will be updated unless `new` is specified.

| name | type | description |
|---|---|---|
| name | string | The name of the flag that will be created or updated. |
| type_id | int | The internal flag type ID that will be created or updated. You will need to specify the `type_id` if more than one flag type of the same name exists. |
| status | string | The flags new status (i.e. "?", "+", "-" or "X" to clear a flag). |
| requestee | string | The login of the requestee if the flag type is requestable to a specific user. |
| id | int | Use ID to specify the flag to be updated. You will need to specify the `id` if more than one flag is set of the same name. |
| new | boolean | Set to true if you specifically want a new flag to be created. |

**Response**

```
{
  "attachments" : [
    {
      "changes" : {
        "content_type" : {
          "added" : "text/xml",
          "removed" : "text/plain"
        },
        "is_patch" : {
          "added" : "0",
          "removed" : "1"
        },
        "summary" : {
          "added" : "Test XML file",
          "removed" : "test patch"
```

(continues on next page)

(continued from previous page)

```
        }
      },
      "id" : 2796,
      "last_change_time" : "2014-09-29T14:41:53Z"
    }
  ]
}
```

`attachments` (array) Change objects with the following items:

| name | type | description |
|---|---|---|
| id | int | The ID of the attachment that was updated. |
| last_change_time | datetime | The exact time that this update was done at, for this attachment. If no update was done (that is, no fields had their values changed and no comment was added) then this will instead be the last time the attachment was updated. |
| changes | object | The changes that were actually done on this attachment. The keys are the names of the fields that were changed, and the values are an object with two items:<br>• added: (string) The values that were added to this field. Possibly a comma-and-space-separated list if multiple values were added.<br>• removed: (string) The values that were removed from this field. |

**Errors**

This method can throw all the same errors as *Get Bug*, plus:

- 129 (Flag Status Invalid) The flag status is invalid.
- 130 (Flag Modification Denied) You tried to request, grant, or deny a flag but only a user with the required permissions may make the change.
- 131 (Flag not Requestable from Specific Person) You can't ask a specific person for the flag.
- 132 (Flag not Unique) The flag specified has been set multiple times. You must specify the id value to update the flag.
- 133 (Flag Type not Unique) The flag type specified matches several flag types. You must specify the type id value to update or add a flag.
- 134 (Inactive Flag Type) The flag type is inactive and cannot be used to create new flags.
- 140 (Markdown Disabled) You tried to set the "is_markdown" flag of the "comment" to true but Markdown feature is not enabled.

- 601 (Invalid MIME Type) You specified a “content_type” argument that was blank, not a valid MIME type, or not a MIME type that Bugzilla accepts for attachments.
- 603 (File Name Not Specified) You did not specify a valid for the “file_name” argument.
- 604 (Summary Required) You did not specify a value for the “summary” argument.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

### 5.1.3 Bugs

The REST API for creating, changing, and getting the details of bugs.

This part of the Bugzilla REST API allows you to file new bugs in Bugzilla and to get information about existing bugs.

#### Get Bug

Gets information about particular bugs in the database.

**Request**

To get information about a particular bug using its ID or alias:

```
GET /rest/bug/(id_or_alias)
```

You can also use *Search Bugs* to return more than one bug at a time by specifying bug IDs as the search terms.

```
GET /rest/bug?id=12434,43421
```

| name | type | description |
|---|---|---|
| **id_or_alias** | mixed | An integer bug ID or a bug alias string. |

**Response**

```
{
  "faults": [],
  "bugs": [
    {
      "assigned_to_detail": {
        "id": 2,
        "real_name": "Test User",
        "nick": "user",
        "name": "user@bugzilla.org",
        "email": "user@bugzilla.org"
      },
      "flags": [
        {
          "type_id": 11,
          "modification_date": "2014-09-28T21:03:47Z",
          "name": "blocker",
          "status": "?",
          "id": 2906,
```

(continues on next page)

(continued from previous page)

```
          "setter": "user@bugzilla.org",
          "creation_date": "2014-09-28T21:03:47Z"
        }
      ],
      "resolution": "INVALID",
      "id": 35,
      "type": "defect",
      "qa_contact": "",
      "triage_owner": "",
      "version": "1.0",
      "status": "RESOLVED",
      "creator": "user@bugzilla.org",
      "cf_drop_down": "---",
      "summary": "test bug",
      "last_change_time": "2014-09-23T19:12:17Z",
      "platform": "All",
      "url": "",
      "classification": "Unclassified",
      "cc_detail": [
        {
          "id": 786,
          "real_name": "Foo Bar",
          "nick": "foo",
          "name": "foo@bar.com",
          "email": "foo@bar.com"
        },
      ],
      "priority": "P1",
      "is_confirmed": true,
      "creation_time": "2000-07-25T13:50:04Z",
      "assigned_to": "user@bugzilla.org",
      "flags": [],
      "alias": null,
      "cf_large_text": "",
      "groups": [],
      "op_sys": "All",
      "cf_bug_id": null,
      "depends_on": [],
      "is_cc_accessible": true,
      "is_open": false,
      "cf_qa_list_4": "---",
      "keywords": [],
      "cc": [
        "foo@bar.com",
      ],
      "see_also": [],
      "deadline": null,
      "is_creator_accessible": true,
      "whiteboard": "",
      "dupe_of": null,
      "duplicates": [],
      "target_milestone": "---",
```

(continues on next page)

(continued from previous page)

```
      "cf_mulitple_select": [],
      "component": "SaltSprinkler",
      "severity": "critical",
      "cf_date": null,
      "product": "FoodReplicator",
      "creator_detail": {
        "id": 28,
        "real_name": "hello",
        "nick": "namachi",
        "name": "user@bugzilla.org",
        "email": "namachi@netscape.com"
      },
      "cf_free_text": "",
      "blocks": [],
      "regressed_by": [],
      "regressions": [],
      "comment_count": 12
    }
  ]
}
```

`bugs` (array) Each bug object contains information about the bugs with valid ids containing the following items:

These fields are returned by default or by specifying `_default` in `include_fields`.

| name | type | description |
|---|---|---|
| actual_time | double | The total number of hours that this bug has taken so far. If you are not in the time-tracking group, |
| alias | string | The unique alias of this bug. A `null` value will be returned if this bug has no alias. |
| assigned_to | string | The login name of the user to whom the bug is assigned. |
| assigned_to_detail | object | An object containing detailed user information for the assigned_to. To see the keys included in the |
| blocks | array | The IDs of bugs that are “blocked” by this bug. |
| cc | array | The login names of users on the CC list of this bug. |
| cc_detail | array | Array of objects containing detailed user information for each of the cc list members. To see the k |
| classification | string | The name of the current classification the bug is in. |
| component | string | The name of the current component of this bug. |
| creation_time | datetime | When the bug was created. |
| creator | string | The login name of the person who filed this bug (the reporter). |
| creator_detail | object | An object containing detailed user information for the creator. To see the keys included in the user |
| deadline | string | The day that this bug is due to be completed, in the format `YYYY-MM-DD`. |
| depends_on | array | The IDs of bugs that this bug “depends on”. |
| dupe_of | int | The bug ID of the bug that this bug is a duplicate of. If this bug isn’t a duplicate of any bug, this w |
| duplicates | array | The ids of bugs that are marked as duplicate of this bug. |
| estimated_time | double | The number of hours that it was estimated that this bug would take. If you are not in the time-track |
| flags | array | An array of objects containing the information about flags currently set for the bug. Each flag obje |
| groups | array | The names of all the groups that this bug is in. |
| id | int | The unique numeric ID of this bug. |
| is_cc_accessible | boolean | If true, this bug can be accessed by members of the CC list, even if they are not in the groups the b |
| is_confirmed | boolean | `true` if the bug has been confirmed. Usually this means that the bug has at some point been move |
| is_open | boolean | `true` if this bug is open, `false` if it is closed. |
| is_creator_accessible | boolean | If `true`, this bug can be accessed by the creator of the bug, even if they are not a member of the g |
| keywords | array | Each keyword that is on this bug. |

Table 1 – continued from previous page

| name | type | description |
|---|---|---|
| last_change_time | datetime | When the bug was last changed. |
| comment_count | int | Number of comments associated with the bug. |
| op_sys | string | The name of the operating system that the bug was filed against. |
| platform | string | The name of the platform (hardware) that the bug was filed against. |
| priority | string | The priority of the bug. |
| product | string | The name of the product this bug is in. |
| qa_contact | string | The login name of the current QA Contact on the bug. |
| qa_contact_detail | object | An object containing detailed user information for the qa_contact. To see the keys included in the |
| regressed_by | array | The IDs of bugs that introduced this bug. |
| regressions | array | The IDs of bugs that are introduced by this bug. |
| remaining_time | double | The number of hours of work remaining until work on this bug is complete. If you are not in the ti |
| resolution | string | The current resolution of the bug, or an empty string if the bug is open. |
| see_also | array | The URLs in the See Also field on the bug. |
| severity | string | The current severity of the bug. |
| status | string | The current status of the bug. |
| summary | string | The summary of this bug. |
| target_milestone | string | The milestone that this bug is supposed to be fixed by, or for closed bugs, the milestone that it was |
| type | string | The type of the bug. |
| update_token | string | The token that you would have to pass to the `process_bug.cgi` page in order to update this bug. |
| url | string | A URL that demonstrates the problem described in the bug, or is somehow related to the bug repo |
| version | string | The version the bug was reported against. |
| whiteboard | string | The value of the “status whiteboard” field on the bug. |

Custom fields:

Every custom field in this installation will also be included in the return value. Most fields are returned as strings. However, some field types have different return values.

Normally custom fields are returned by default similar to normal bug fields or you can specify only custom fields by using `_custom` in `include_fields`.

Extra fields:

These fields are returned only by specifying `_extra` or the field name in `include_fields`.

| name | type | description |
|---|---|---|
| attachments | array | Each array item is an Attachment object. See *Get Attachment* for details of the object. |
| comments | array | Each array item is a Comment object. See *Get Comments* for details of the object. |
| counts | object | An object containing the numbers of the items in the following fields: `attachments`, `cc`, `comments`, `keywords`, `blocks`, `depends_on`, `regressed_by`, `regressions` and `duplicates`. |
| description | string | The description (initial comment) of the bug. |
| filed_via | string | How the bug was filed, e.g. `standard_form`. |
| history | array | Each array item is a History object. See *Bug History* for details of the object. |
| tags | array | Each array item is a tag name. Note that tags are personal to the currently logged in user and are not the same as comment tags. |
| triage_owner | string | The login name of the Triage Owner of the bug’s component. |
| triage_owner_detail | object | An object containing detailed user information for the `triage_owner`. To see the keys included in the user detail object, see below. |

User object:

| name | type | description |
|---|---|---|
| id | int | The user ID for this user. |
| real_name | string | The 'real' name for this user, if any. |
| nick | string | The user's nickname. Currently this is extracted from the real_name, name or email field. |
| name | string | The user's Bugzilla login. |
| email | string | The user's email address. Currently this is the same value as the name. |

Flag object:

| name | type | description |
|---|---|---|
| id | int | The ID of the flag. |
| name | string | The name of the flag. |
| type_id | int | The type ID of the flag. |
| creation_date | datetime | The timestamp when this flag was originally created. |
| modification_date | datetime | The timestamp when the flag was last modified. |
| status | string | The current status of the flag. |
| setter | string | The login name of the user who created or last modified the flag. |
| requestee | string | The login name of the user this flag has been requested to be granted or denied. Note, this field is only returned if a requestee is set. |

Custom field object:

You can specify to only return custom fields by specifying `_custom` or the field name in `include_fields`.

- Bug ID Fields: (int)
- Multiple-Selection Fields: (array of strings)
- Date/Time Fields: (datetime)

**Errors**

- 100 (Invalid Bug Alias) If you specified an alias and there is no bug with that alias.
- 101 (Invalid Bug ID) The bug_id you specified doesn't exist in the database.
- 102 (Access Denied) You do not have access to the bug_id you specified.

## Bug History

Gets the history of changes for particular bugs in the database.

**Request**

To get the history for a specific bug ID:

```
GET /rest/bug/(id)/history
```

To get the history for a bug since a specific date:

```
GET /rest/bug/(id)/history?new_since=YYYY-MM-DD
```

| name | type | description |
|---|---|---|
| **id** | mixed | An integer bug ID or alias. |
| new_since | datetime | A datetime timestamp to only show history since. |

**Response**

```
{
  "bugs": [
    {
      "alias": null,
      "history": [
        {
          "when": "2014-09-23T19:12:17Z",
          "who": "user@bugzilla.org",
          "changes": [
            {
              "added": "P1",
              "field_name": "priority",
              "removed": "P2"
            },
            {
              "removed": "blocker",
              "field_name": "severity",
              "added": "critical"
            }
          ]
        },
        {
          "when": "2014-09-28T21:03:47Z",
          "who": "user@bugzilla.org",
          "changes": [
            {
              "added": "blocker?",
              "removed": "",
              "field_name": "flagtypes.name"
            }
          ]
        }
      ],
      "id": 35
    }
  ]
}
```

`bugs` (array) Bug objects each containing the following items:

| name | type | description |
|---|---|---|
| id | int | The numeric ID of the bug. |
| alias | string | The unique alias of this bug. A `null` value will be returned if this bug has no alias. |
| history | array | An array of History objects. |

History object:

| name | type | description |
|---|---|---|
| when | date-time | The date the bug activity/change happened. |
| who | string | The login name of the user who performed the bug change. |
| changes | array | An array of Change objects which contain all the changes that happened to the bug at this time (as specified by `when`). |

Change object:

| name | type | description |
|---|---|---|
| field_name | string | The name of the bug field that has changed. |
| removed | string | The previous value of the bug field which has been deleted by the change. |
| added | string | The new value of the bug field which has been added by the change. |
| attachment_id | int | The ID of the attachment that was changed. This only appears if the change was to an attachment, otherwise `attachment_id` will not be present in this object. |

**Errors**

Same as *Get Bug*.

### Search Bugs

Allows you to search for bugs based on particular criteria.

**Request**

To search for bugs:

```
GET /rest/bug
```

Unless otherwise specified in the description of a parameter, bugs are returned if they match *exactly* the criteria you specify in these parameters. That is, we don't match against substrings–if a bug is in the "Widgets" product and you ask for bugs in the "Widg" product, you won't get anything.

Criteria are joined in a logical AND. That is, you will be returned bugs that match *all* of the criteria, not bugs that match *any* of the criteria.

Each parameter can be either the type it says, or a list of the types it says. If you pass an array, it means "Give me bugs with *any* of these values." For example, if you wanted bugs that were in either the "Foo" or "Bar" products, you'd pass:

```
GET /rest/bug?product=Foo&product=Bar
```

Some Bugzillas may treat your arguments case-sensitively, depending on what database system they are using. Most commonly, though, Bugzilla is not case-sensitive with the arguments passed (because MySQL is the most-common database to use with Bugzilla, and MySQL is not case sensitive).

In addition to the fields listed below, you may also use criteria that is similar to what is used in the Advanced Search screen of the Bugzilla UI. This includes fields specified by `Search by Change History` and `Custom Search`. The easiest way to determine what the field names are and what format Bugzilla expects is to first construct your query using the Advanced Search UI, execute it and use the query parameters in they URL as your query for the REST call.

| name | type | description |
|---|---|---|
| alias | string | The unique alias of this bug. A `null` value will be returned if this bug has no alias. |
| assigned_to | string | The login name of a user that a bug is assigned to. |
| component | string | The name of the Component that the bug is in. Note that if there are multiple Components with the sa |
| count_only | boolean | If set to true, an object with a single key called "bug_count" will be returned which is the number of b |
| creation_time | datetime | Searches for bugs that were created at this time or later. May not be an array. |
| creator | string | The login name of the user who created the bug. You can also pass this argument with the name `repo` |
| description | string | The description (initial comment) of the bug. |
| filed_via | string | Searches for bugs that were created with this method. |
| id | int | The numeric ID of the bug. |
| last_change_time | datetime | Searches for bugs that were modified at this time or later. May not be an array. |
| limit | int | Limit the number of results returned. If the value is unset, zero or greater than the maximum value set |
| longdescs.count | int | The number of comments a bug has. The bug's description is the first comment. For example, to find |
| offset | int | Used in conjunction with the `limit` argument, `offset` defines the starting position for the search. Fo |
| op_sys | string | The "Operating System" field of a bug. |
| platform | string | The Platform (sometimes called "Hardware") field of a bug. |
| priority | string | The Priority field on a bug. |
| product | string | The name of the Product that the bug is in. |
| quicksearch | string | Search for bugs using quicksearch syntax. |
| resolution | string | The current resolution–only set if a bug is closed. You can find open bugs by searching for bugs with a |
| severity | string | The Severity field on a bug. |
| status | string | The current status of a bug (not including its resolution, if it has one, which is a separate field above). |
| summary | string | Searches for substrings in the single-line Summary field on bugs. If you specify an array, then bugs w |
| tags | string | Searches for a bug with the specified tag. If you specify an array, then any bugs that match *any* of the |
| target_milestone | string | The Target Milestone field of a bug. Note that even if this Bugzilla does not have the Target Milestone |
| qa_contact | string | The login name of the bug's QA Contact. Note that even if this Bugzilla does not have the QA Contac |
| triage_owner | string | The login name of the Triage Owner of a bug's component. |
| type | string | The Type field on a bug. |
| url | string | The "URL" field of a bug. |
| version | string | The Version field of a bug. |
| whiteboard | string | Search the "Status Whiteboard" field on bugs for a substring. Works the same as the `summary` field de |

**Response**

The same as *Get Bug*.

**Errors**

If you specify an invalid value for a particular field, you just won't get any results for that value.

- 1000 (Parameters Required) You may not search without any search terms.

### Create Bug

This allows you to create a new bug in Bugzilla. If you specify any invalid fields, an error will be thrown stating which field is invalid. If you specify any fields you are not allowed to set, they will just be set to their defaults or ignored.

You cannot currently set all the items here that you can set on enter_bug.cgi.

The WebService interface may allow you to set things other than those listed here, but realize that anything undocumented here may likely change in the future.

**Request**

To create a new bug in Bugzilla.

```
POST /rest/bug
```

```
{
  "product" : "TestProduct",
  "component" : "TestComponent",
  "version" : "unspecified",
  "summary" : "'This is a test bug - please disregard",
  "alias" : "SomeAlias",
  "op_sys" : "All",
  "priority" : "P1",
  "rep_platform" : "All"
}
```

Some params must be set, or an error will be thrown. These params are marked in **bold**.

Some parameters can have defaults set in Bugzilla, by the administrator. If these parameters have defaults set, you can omit them. These parameters are marked (defaulted).

Clients that want to be able to interact uniformly with multiple Bugzillas should always set both the params marked required and those marked (defaulted), because some Bugzillas may not have defaults set for (defaulted) parameters, and then this method will throw an error if you don't specify them.

| name | type | description |
|---|---|---|
| **product** | string | The name of the product the bug is being filed against. |
| **component** | string | The name of a component in the product above. |
| **summary** | string | A brief description of the bug being filed. |
| **version** | string | A version of the product above; the version the bug was found in. |
| description | string | (defaulted) The description (initial comment) of the bug. Some Bugzilla installations require this to not be blank. |
| filed_via | string | (defaulted) How the bug is being filed. It will be `api` by default when filing through the API. |
| op_sys | string | (defaulted) The operating system the bug was discovered on. |
| platform | string | (defaulted) What type of hardware the bug was experienced on. |
| priority | string | (defaulted) What order the bug will be fixed in by the developer, compared to the developer's other bugs. |
| severity | string | (defaulted) How severe the bug is. |
| type | string | (defaulted) The basic category of the bug. Some Bugzilla installations require this to be specified. |
| alias | string | The alias for the bug that can be used instead of a bug number when accessing this bug. Must be unique in all of this Bugzilla. |
| assigned_to | string | A user to assign this bug to, if you don't want it to be assigned to the component owner. |
| cc | array | An array of usernames to CC on this bug. |
| comment_is_private | boolean | If set to true, the description is private, otherwise it is assumed to be public. |
| groups | array | An array of group names to put this bug into. You can see valid group names on the Permissions tab of the Preferences screen, or, if you are an administrator, in the Groups control panel. If you don't specify this argument, then the bug will be added into all the groups that are set as being "Default" for this product. (If you want to avoid that, you should specify `groups` as an empty array.) |
| qa_contact | string | If this installation has QA Contacts enabled, you can set the QA Contact here if you don't want to use the component's default QA Contact. |
| status | string | The status that this bug should start out as. Note that only certain statuses can be set on bug creation. |
| resolution | string | If you are filing a closed bug, then you will have to specify a resolution. You cannot currently specify a resolution of `DUPLICATE` for new bugs, though. That must be done with *Update Bug*. |
| target_milestone | string | A valid target milestone for this product. |
| flags | array | Flags objects to add to the bug. The object format is described in the Flag object below. |
| keywords | array | One or more valid keywords to add to this bug. |
| dependson | array | One or more valid bug ids that this bug depends on. |
| blocked | array | One or more valid bug ids that this bug blocks. |
| regressed_by | array | One or more valid bug ids that introduced this bug. |

Flag object:

To create a flag, at least the `status` and the `type_id` or `name` must be provided. An optional requestee can be passed if the flag type is requestable to a specific user.

| name | type | description |
|---|---|---|
| name | string | The name of the flag type. |
| type_id | int | The internal flag type ID. |
| status | string | The flags new status (i.e. "?", "+", "-" or "X" to clear flag). |
| requestee | string | The login of the requestee if the flag type is requestable to a specific user. |

In addition to the above parameters, if your installation has any custom fields, you can set them just by passing in the name of the field and its value as a string.

**Response**

```
{
  "id" : 12345
}
```

| name | type | description |
|---|---|---|
| id | int | This is the ID of the newly-filed bug. |

**Errors**

- 51 (Invalid Object) You specified a field value that is invalid. The error message will have more details.
- 103 (Invalid Alias) The alias you specified is invalid for some reason. See the error message for more details.
- 104 (Invalid Field) One of the drop-down fields has an invalid value, or a value entered in a text field is too long. The error message will have more detail.
- 105 (Invalid Component) You didn't specify a component.
- 106 (Invalid Product) Either you didn't specify a product, this product doesn't exist, or you don't have permission to enter bugs in this product.
- 107 (Invalid Summary) You didn't specify a summary for the bug.
- 116 (Dependency Loop) You specified values in the "blocks" and "depends_on" fields, or the "regressions" and "regressed_by" fields, that would cause a circular dependency between bugs.
- 120 (Group Restriction Denied) You tried to restrict the bug to a group which does not exist, or which you cannot use with this product.
- 129 (Flag Status Invalid) The flag status is invalid.
- 130 (Flag Modification Denied) You tried to request, grant, or deny a flag but only a user with the required permissions may make the change.
- 131 (Flag not Requestable from Specific Person) You can't ask a specific person for the flag.
- 133 (Flag Type not Unique) The flag type specified matches several flag types. You must specify the type id value to update or add a flag.
- 134 (Inactive Flag Type) The flag type is inactive and cannot be used to create new flags.
- 135 (Bug Type Required) You didn't specify a type for the bug.
- 504 (Invalid User) Either the QA Contact, Assignee, or CC lists have some invalid user in them. The error message will have more details.

## Update Bug

Allows you to update the fields of a bug. Automatically sends emails out about the changes.

**Request**

To update the fields of a current bug.

```
PUT /rest/bug/(id_or_alias)
```

```
{
  "ids" : [35],
  "status" : "IN_PROGRESS",
  "keywords" : {
    "add" : ["funny", "stupid"]
  }
}
```

The params to include in the PUT body as well as the returned data format, are the same as below. You can specify the ID or alias of the bug to update either in the URL path and/or in the `ids` param. You can use both and they will be combined so you can edit more than one bug at a time.

| name | type | description |
|---|---|---|
| **id_or_alias** | mixed | An integer bug ID or alias. |
| **ids** | array | The IDs or aliases of the bugs that you want to modify. |

All following fields specify the values you want to set on the bugs you are updating.

| name | type | description |
|---|---|---|
| alias | string | The alias for the bug that can be used instead of a bug number when accessing this bug. Must be unique in all of this Bugzilla. |
| assigned_to | string | The full login name of the user this bug is assigned to. |
| blocks | object | (Same as `regressed_by` below) |
| depends_on | object | (Same as `regressed_by` below) |
| regressions | object | (Same as `regressed_by` below) |

continues on next page

Table 3 – continued from previous page

| name | type | description |
|---|---|---|
| regressed_by | object | These specify the bugs that this bug blocks, depends on, regresses, or is regressed by, respectively. To set these, you should pass an object as the value. The object may contain the following items:<br>• `add` (array) Bug IDs to add to this field.<br>• `remove` (array) Bug IDs to remove from this field. If the bug IDs are not already in the field, they will be ignored.<br>• `set` (array of) An exact set of bug IDs to set this field to, overriding the current value. If you specify `set`, then `add` and `remove` will be ignored. |
| cc | object | The users on the cc list. To modify this field, pass an object, which may have the following items:<br>• `add` (array) User names to add to the CC list. They must be full user names, and an error will be thrown if you pass in an invalid user name.<br>• `remove` (array) User names to remove from the CC list. They must be full user names, and an error will be thrown if you pass in an invalid user name. |
| is_cc_accessible | boolean | Whether or not users in the CC list are allowed to access the bug, even if they aren't in a group that can normally access the bug. |
| comment | object | A comment on the change. The object may contain the following items:<br>• `body` (string) The actual text of the comment. For compatibility with the parameters to *Create Comments*, you can also call this field `comment`, if you want.<br>• `is_private` (boolean) Whether the comment is private or not. If you try to make a comment private and you don't have the permission to, an error will be thrown. |

continues on next page

Table 3 – continued from previous page

| name | type | description |
|---|---|---|
| comment_is_private | object | This is how you update the privacy of comments that are already on a bug. This is a object, where the keys are the `int` ID of comments (not their count on a bug, like #1, #2, #3, but their globally-unique ID, as returned by *Get Comments* and the value is a `boolean` which specifies whether that comment should become private (`true`) or public (`false`).<br>The comment IDs must be valid for the bug being updated. Thus, it is not practical to use this while updating multiple bugs at once, as a single comment ID will never be valid on multiple bugs. |
| component | string | The Component the bug is in. |
| deadline | date | The Deadline field is a date specifying when the bug must be completed by, in the format `YYYY-MM-DD`. |
| dupe_of | int | The bug that this bug is a duplicate of. If you want to mark a bug as a duplicate, the safest thing to do is to set this value and *not* set the `status` or `resolution` fields. They will automatically be set by Bugzilla to the appropriate values for duplicate bugs. |
| estimated_time | double | The total estimate of time required to fix the bug, in hours. This is the *total* estimate, not the amount of time remaining to fix it. |
| flags | array | An array of Flag change objects. The items needed are described below. |

continues on next page

Table 3 – continued from previous page

| name | type | description |
|---|---|---|
| groups | object | The groups a bug is in. To modify this field, pass an object, which may have the following items:<br>• `add` (array) The names of groups to add. Passing in an invalid group name or a group that you cannot add to this bug will cause an error to be thrown.<br>• `remove` (array) The names of groups to remove. Passing in an invalid group name or a group that you cannot remove from this bug will cause an error to be thrown. |
| keywords | object | Keywords on the bug. To modify this field, pass an object, which may have the following items:<br>• `add` (array) The names of keywords to add to the field on the bug. Passing something that isn't a valid keyword name will cause an error to be thrown.<br>• `remove` (array) The names of keywords to remove from the field on the bug. Passing something that isn't a valid keyword name will cause an error to be thrown.<br>• `set` (array) An exact set of keywords to set the field to, on the bug. Passing something that isn't a valid keyword name will cause an error to be thrown. Specifying `set` overrides `add` and `remove`. |
| op_sys | string | The Operating System ("OS") field on the bug. |
| platform | string | The Platform or "Hardware" field on the bug. |
| priority | string | The Priority field on the bug. |

continues on next page

Table 3 – continued from previous page

| name | type | description |
|---|---|---|
| product | string | The name of the product that the bug is in. If you change this, you will probably also want to change `target_milestone`, `version`, and `component`, since those have different legal values in every product.<br>If you cannot change the `target_milestone` field, it will be reset to the default for the product, when you move a bug to a new product.<br>You may also wish to add or remove groups, as which groups are valid on a bug depends on the product. Groups that are not valid in the new product will be automatically removed, and groups which are mandatory in the new product will be automatically added, but no other automatic group changes will be done.<br>**Note:** Users can only move a bug into a product if they would normally have permission to file new bugs in that product. |
| qa_contact | string | The full login name of the bug's QA Contact. |
| is_creator_accessible | boolean | Whether or not the bug's reporter is allowed to access the bug, even if they aren't in a group that can normally access the bug. |
| remaining_time | double | How much work time is remaining to fix the bug, in hours. If you set `work_time` but don't explicitly set `remaining_time`, then the `work_time` will be deducted from the bug's `remaining_time`. |
| reset_assigned_to | boolean | If true, the `assigned_to` field will be reset to the default for the component that the bug is in. (If you have set the component at the same time as using this, then the component used will be the new component, not the old one.) |

continues on next page

Table 3 – continued from previous page

| name | type | description |
|---|---|---|
| reset_qa_contact | boolean | If true, the `qa_contact` field will be reset to the default for the component that the bug is in. (If you have set the component at the same time as using this, then the component used will be the new component, not the old one.) |
| resolution | string | The current resolution. May only be set if you are closing a bug or if you are modifying an already-closed bug. Attempting to set the resolution to *any* value (even an empty or null string) on an open bug will cause an error to be thrown.<br><br>**Note:** If you change the `status` field to an open status, the resolution field will automatically be cleared, so you don't have to clear it manually. |
| see_also | object | The See Also field on a bug, specifying URLs to bugs in other bug trackers. To modify this field, pass an object, which may have the following items:<br>• `add` (array) URLs to add to the field. Each URL must be a valid URL to a bug-tracker, or an error will be thrown.<br>• `remove` (array) URLs to remove from the field. Invalid URLs will be ignored. |
| severity | string | The Severity field of a bug. |
| status | string | The status you want to change the bug to. Note that if a bug is changing from open to closed, you should also specify a `resolution`. |
| summary | string | The Summary field of the bug. |
| target_milestone | string | The bug's Target Milestone. |
| type | string | The Type field on the bug. |
| url | string | The "URL" field of a bug. |
| version | string | The bug's Version field. |
| whiteboard | string | The Status Whiteboard field of a bug. |

continues on next page

Table 3 – continued from previous page

| name | type | description |
|---|---|---|
| work_time | double | The number of hours worked on this bug as part of this change. If you set `work_time` but don't explicitly set `remaining_time`, then the `work_time` will be deducted from the bug's `remaining_time`. |

You can also set the value of any custom field by passing its name as a parameter, and the value to set the field to. For multiple-selection fields, the value should be an array of strings.

Flag change object:

The following values can be specified. At least the `status` and one of `type_id`, `id`, or `name` must be specified. If a `type_id` or `name` matches a single currently set flag, the flag will be updated unless `new` is specified.

| name | type | description |
|---|---|---|
| name | string | The name of the flag that will be created or updated. |
| type_id | int | The internal flag type ID that will be created or updated. You will need to specify the `type_id` if more than one flag type of the same name exists. |
| **status** | string | The flags new status (i.e. "?", "+", "-" or "X" to clear a flag). |
| requestee | string | The login of the requestee if the flag type is requestable to a specific user. |
| id | int | Use ID to specify the flag to be updated. You will need to specify the `id` if more than one flag is set of the same name. |
| new | boolean | Set to true if you specifically want a new flag to be created. |

**Response**

```
{
  "bugs" : [
    {
      "alias" : null,
      "changes" : {
        "keywords" : {
          "added" : "funny, stupid",
          "removed" : ""
        },
          "status" : {
            "added" : "IN_PROGRESS",
            "removed" : "CONFIRMED"
        }
      },
      "id" : 35,
      "last_change_time" : "2014-09-29T14:25:35Z"
    }
  ]
}
```

`bugs` (array) This points to an array of objects with the following items:

| name | type | description |
|---|---|---|
| id | int | The ID of the bug that was updated. |
| alias | string | The alias of the bug that was updated, if this bug has any alias. |
| last_change_time | datetime | The exact time that this update was done at, for this bug. If no update was done (that is, no fields had their values changed and no comment was added) then this will instead be the last time the bug was updated. |
| changes | object | The changes that were actually done on this bug. The keys are the names of the fields that were changed, and the values are an object with two keys:<br>• `added` (string) The values that were added to this field, possibly a comma-and-space-separated list if multiple values were added.<br>• `removed` (string) The values that were removed from this field, possibly a comma-and-space-separated list if multiple values were removed. |

Currently, some fields are not tracked in changes: `comment`, `comment_is_private`, and `work_time`. This means that they will not show up in the return value even if they were successfully updated. This may change in a future version of Bugzilla.

**Errors**

This method can throw all the same errors as *Get Bug*, plus:

- 129 (Flag Status Invalid) The flag status is invalid.
- 130 (Flag Modification Denied) You tried to request, grant, or deny a flag but only a user with the required permissions may make the change.
- 131 (Flag not Requestable from Specific Person) You can't ask a specific person for the flag.
- 132 (Flag not Unique) The flag specified has been set multiple times. You must specify the id value to update the flag.
- 133 (Flag Type not Unique) The flag type specified matches several flag types. You must specify the type id value to update or add a flag.
- 134 (Inactive Flag Type) The flag type is inactive and cannot be used to create new flags.
- 140 (Markdown Disabled) You tried to set the "is_markdown" flag of the "comment" to true but Markdown feature is not enabled.
- 601 (Invalid MIME Type) You specified a "content_type" argument that was blank, not a valid MIME type, or not a MIME type that Bugzilla accepts for attachments.
- 603 (File Name Not Specified) You did not specify a valid for the "file_name" argument.
- 604 (Summary Required) You did not specify a value for the "summary" argument.

### Possible Duplicates

Gets a list of possible duplicate bugs.

**Request**

To search by similar bug.

```
GET /rest/bug/possible_duplicates?id=1234567
```

To search by a similar bug summary directly.

```
GET /rest/bug/possible_duplicates?summary=Similar+Bug+Summary
```

| name | type | description |
|---|---|---|
| id | int | The id of a bug to find duplicates of. |
| summary | string | A summary to search for duplicates of, only used if no bug id is given. |
| product | string | A product group to limit the search in. |
| limit | int | Limit the number of results returned. If the value is unset, zero or greater than the maximum value set by the administrator, which is 10,000 by default, then the maximum value will be used instead. This is a preventive measure against DoS-like attacks on Bugzilla. |

**Response**

```
{
  "bugs": [
    {
      "alias": null,
      "history": [
        {
          "when": "2014-09-23T19:12:17Z",
          "who": "user@bugzilla.org",
          "changes": [
            {
              "added": "P1",
              "field_name": "priority",
              "removed": "P2"
            },
            {
              "removed": "blocker",
              "field_name": "severity",
              "added": "critical"
            }
          ]
        },
        {
          "when": "2014-09-28T21:03:47Z",
          "who": "user@bugzilla.org",
          "changes": [
            {
              "added": "blocker?",
              "removed": "",
```

(continues on next page)

(continued from previous page)

```
                "field_name": "flagtypes.name"
              }
            ]
          }
        ],
        "id": 35
      }
    ]
}
```

`bugs` (array) Bug objects each containing the following items. If a bug id was used to query this endpoint, that bug will not be in the list returned.

| name | type | description |
|---|---|---|
| id | int | The numeric ID of the bug. |
| alias | string | The unique alias of this bug. A `null` value will be returned if this bug has no alias. |
| history | array | An array of History objects. |

History object:

| name | type | description |
|---|---|---|
| when | date-time | The date the bug activity/change happened. |
| who | string | The login name of the user who performed the bug change. |
| changes | array | An array of Change objects which contain all the changes that happened to the bug at this time (as specified by `when`). |

Change object:

| name | type | description |
|---|---|---|
| field_name | string | The name of the bug field that has changed. |
| removed | string | The previous value of the bug field which has been deleted by the change. |
| added | string | The new value of the bug field which has been added by the change. |
| attachment_id | int | The ID of the attachment that was changed. This only appears if the change was to an attachment, otherwise `attachment_id` will not be present in this object. |

---

This documentation undoubtedly has bugs; if you find some, please file them here.

### 5.1.4 Bug User Last Visited

#### Update Last Visited

Update the last-visited time for the specified bug and current user.

**Request**

To update the time for a single bug id:

```
POST /rest/bug_user_last_visit/(id)
```

To update one or more bug ids at once:

```
POST /rest/bug_user_last_visit
```

```
{
  "ids" : [35,36,37]
}
```

| name | type | description |
|---|---|---|
| **id** | int | An integer bug id. |
| **ids** | array | One or more bug ids to update. |

**Response**

```
[
  {
    "id" : 100,
    "last_visit_ts" : "2014-10-16T17:38:24Z"
  }
]
```

An array of objects containing the items:

| name | type | description |
|---|---|---|
| id | int | The bug id. |
| last_visit_ts | datetime | The timestamp the user last visited the bug. |

**Errors**

- 1300 (User Not Involved with Bug) The caller's account is not involved with the bug id provided.

### Get Last Visited

**Request**

Get the last-visited timestamp for one or more specified bug ids or get a list of the last 20 visited bugs and their timestamps.

To return the last-visited timestamp for a single bug id:

```
GET /rest/bug_user_last_visit/(id)
```

To return more than one specific bug timestamps:

```
GET /rest/bug_user_last_visit/123?ids=234&ids=456
```

To return all the timestamps stored during the retention period:

```
GET /rest/bug_user_last_visit
```

| name | type | description |
|---|---|---|
| **id** | int | An integer bug id. |
| **ids** | array | One or more optional bug ids to get. |

**Response**

```
[
  {
    "id" : 100,
    "last_visit_ts" : "2014-10-16T17:38:24Z"
  }
]
```

An array of objects containing the following items:

| name | type | description |
|---|---|---|
| id | int | The bug id. |
| last_visit_ts | datetime | The timestamp the user last visited the bug. |

---

This documentation undoubtedly has bugs; if you find some, please file them here.

### 5.1.5 Flag Activity

This API provides information about activity relating to bug and attachment flags.

#### Get Flag Activity

**Request**

There are a variety of methods for querying flag activity based on different criteria.

```
GET /rest/review/flag_activity/(flag_id)
```

Fetches activity for the given flag as specified by its id.

```
GET /rest/review/flag_activity/requestee/(requestee)
```

Fetches activity for flags where the requestee matches the given Bugzilla login.

```
GET /rest/review/flag_activity/setter/(requestee)
```

Fetches activity for flags where the setter matches the given Bugzilla login.

```
GET /rest/review/flag_activity/type_id/(type_id)
```

Fetches activity for all flags of the type specified by its id.

```
GET /rest/review/flag_activity/type_name/(type_name)
```

Fetches activity for all flags of the type specified by its name.

```
GET /rest/review/flag_activity
```

Fetches activity for all flags.

There are also query parameters that can be used to further filter the response:

| name | type | description |
|---|---|---|
| limit | int | Number of entries to return. |
| offset | int | Number of entries to skip before returning results. |
| after | date | Display activity occurring on or after this date. |
| before | date | Display activity occurring before this date. |

Note that if `offset` is specified, `limit` must be given as well.

There is a site-specific maximum number of entries that will be returned regardless of the value given for `limit`. This is also the default if `limit` is not specified.

For example, to get the first 100 flag-activity entries that occurred on or after 2018-01-01 for flag ID 42:

```
GET /rest/review/flag_activity/42?limit=100&after=2018-01-01
```

**Response**

```
[
  {
    "attachment_id": null,
    "bug_id": 1395127,
    "creation_time": "2018-10-10 12:41:00",
    "flag_id": 1637223,
    "id": 1449303,
    "requestee": {
      "id": 123,
      "name": "user@mozilla.com",
      "nick": "user",
      "real_name": "J. Random User"
    },
    "setter": {
      "id": 123,
      "name": "user@mozilla.com",
      "nick": "user",
      "real_name": "J. Random User"
    },
    "status": "?",
    "type": {
      "description": "Set this flag when the bug is in need of additional information.",
      "id": 800,
      "is_active": true,
      "is_multiplicable": true,
      "is_requesteeble": true,
      "name": "needinfo",
      "type": "bug"
    }
  }
]
```

An object containing a list of flags. The fields for each flag are as follows:

| name | type | description |
|---|---|---|
| attachment_id | int | The numeric ID of the associated attachment, if any. |
| bug_id | int | The numeric ID of the associated bug. |
| creation_time | datetime | The time the flag status changed. |
| flag_id | int | The numeric ID of this flag instance. |
| id | int | The numeric ID of this flag-activity event. |
| requestee | object | Data about the user of which the flag was requested. |
| setter | object | Data about the user who set the flag. |
| status | string | Status of the flag: "?", "+", or "-". |
| type | object | Data about the type of flag. |

The requestee and setter objects have the following fields:

| name | type | description |
|---|---|---|
| id | int | The unique ID of the user. |
| name | string | The login of the user (typically an email address). |
| real_name | string | The real name of the user, if set. |
| nick | string | The user's nickname. Currently this is extracted the real_name, name or email field. |

The type object has the following fields:

| name | type | description |
|---|---|---|
| description | string | A plain-English description of the flag type. |
| id | int | The numeric ID of the flag type. |
| is_active | boolean | Indicates if the flag type can be used. |
| is_multiplicable | boolean | Indicates if more than one flags of this type can be set on a bug/attachment. |
| is_requesteeble | boolean | Indicates if this flag type supports a requestee. |
| name | string | Short descriptive name of this flag type. |
| type | string | The object to which this flag type can be applied (e.g. "bug", "attachment"). |

**Errors**

If a nonexistent but properly specified (i.e. integer value) flag or flag-type ID is given, a 200 OK response will be returned with an empty array. In other cases, different response codes may be returned:

- 400 (Bad Request): An invalid flag or flag-type ID was given, or `offset` was given without a value for `limit`.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

### 5.1.6 Bugzilla Information

These methods are used to get general configuration information about this Bugzilla instance.

### Version

Returns the current version of Bugzilla. Normally in the format of `X.X` or `X.X.X`. For example, `4.4` for the initial release of a new branch. Or `4.4.6` for a minor release on the same branch.

**Request**

```
GET /rest/version
```

**Response**

```
{
  "version": "4.5.5+"
}
```

| name | type | description |
|---|---|---|
| version | string | The current version of this Bugzilla |

### Extensions

Gets information about the extensions that are currently installed and enabled in this Bugzilla.

**Request**

```
GET /rest/extensions
```

**Response**

```
{
  "extensions": {
    "Voting": {
      "version": "4.5.5+"
    },
    "BmpConvert": {
      "version": "1.0"
    }
  }
}
```

| name | type | description |
|---|---|---|
| extensions | object | An object containing the extensions enabled as keys. Each extension object contains the following keys: • `version` (string) The version of the extension. |

### Timezone

Returns the timezone in which Bugzilla expects to receive dates and times on the API. Currently hard-coded to UTC ("+0000"). This is unlikely to change.

**Request**

```
GET /rest/timezone
```

```
{
  "timezone": "+0000"
}
```

**Response**

| name | type | description |
|---|---|---|
| timezone | string | The timezone offset as a string in (+/-)XXXX (RFC 2822) format. |

### Time

Gets information about what time the Bugzilla server thinks it is, and what timezone it's running in.

**Request**

```
GET /rest/time
```

**Response**

```
{
  "web_time_utc": "2014-09-26T18:01:30Z",
  "db_time": "2014-09-26T18:01:30Z",
  "web_time": "2014-09-26T18:01:30Z",
  "tz_offset": "+0000",
  "tz_short_name": "UTC",
  "tz_name": "UTC"
}
```

| name | type | description |
|---|---|---|
| db_time | string | The current time in UTC, according to the Bugzilla database server.<br>Note that Bugzilla assumes that the database and the webserver are running in the same time zone. However, if the web server and the database server aren't synchronized or some reason, *this* is the time that you should rely on or doing searches and other input to the WebService. |
| web_time | string | This is the current time in UTC, according to Bugzilla's web server.<br>This might be different by a second from `db_time` since this comes from a different source. If it's any more different than a second, then there is likely some problem with this Bugzilla instance. In this case you should rely on the `db_time`, not the `web_time`. |
| web_time_utc | string | Identical to `web_time`. (Exists only for backwards-compatibility with versions of Bugzilla before 3.6.) |
| tz_name | string | The literal string `UTC`. (Exists only for backwards-compatibility with versions of Bugzilla before 3.6.) |
| tz_short_name | string | The literal string `UTC`. (Exists only for backwards-compatibility with versions of Bugzilla before 3.6.) |
| tz_offset | string | The literal string `+0000`. (Exists only for backwards-compatibility with versions of Bugzilla before 3.6.) |

## Parameters

Returns parameter values currently used in this Bugzilla.

**Request**

```
GET /rest/parameters
```

**Response**

Example response for anonymous user:

```
{
   "parameters" : {
      "maintainer" : "admin@example.com",
      "requirelogin" : "0"
   }
}
```

Example response for authenticated user:

```
{
   "parameters" : {
       "allowemailchange" : "1",
       "attachment_base" : "http://bugzilla.example.com/",
       "commentonchange_resolution" : "0",
       "commentonduplicate" : "0",
       "cookiepath" : "/",
       "createemailregexp" : ".*",
       "defaultopsys" : "",
       "defaultplatform" : "",
       "defaultpriority" : "--",
       "defaultseverity" : "normal",
       "default_bug_type" : "--",
       "duplicate_or_move_bug_status" : "RESOLVED",
       "emailregexp" : "^[\\w\\.\\+\\-=']+@[\\w\\.\\-]+\\.[\\w\\-]+$",
       "emailsuffix" : "",
       "letsubmitterchoosemilestone" : "1",
       "letsubmitterchoosepriority" : "1",
       "mailfrom" : "bugzilla-daemon@example.com",
       "maintainer" : "admin@example.com",
       "maxattachmentsize" : "1000",
       "maxlocalattachment" : "0",
       "musthavemilestoneonaccept" : "0",
       "noresolveonopenblockers" : "0",
       "password_complexity" : "no_constraints",
       "rememberlogin" : "on",
       "require_bug_type" : "1",
       "requirelogin" : "0",
       "urlbase" : "http://bugzilla.example.com/",
       "use_regression_fields" : "1",
       "use_see_also" : "1",
       "useclassification" : "1",
       "usemenuforusers" : "0",
       "useqacontact" : "1",
```

(continues on next page)

(continued from previous page)

```
        "usestatuswhiteboard" : "1",
        "usetargetmilestone" : "1",
    }
}
```

A logged-out user can only access the `maintainer` and `requirelogin` parameters.

A logged-in user can access the following parameters (listed alphabetically):

- allowemailchange
- attachment_base
- commentonchange_resolution
- commentonduplicate
- cookiepath
- defaultopsys
- defaultplatform
- defaultpriority
- defaultseverity
- default_bug_type
- duplicate_or_move_bug_status
- emailregexpdesc
- emailsuffix
- letsubmitterchoosemilestone
- letsubmitterchoosepriority
- mailfrom
- maintainer
- maxattachmentsize
- maxlocalattachment
- musthavemilestoneonaccept
- noresolveonopenblockers
- password_complexity
- rememberlogin
- require_bug_type
- requirelogin
- search_allow_no_criteria
- urlbase
- use_regression_fields
- use_see_also
- useclassification

- usemenuforusers
- useqacontact
- usestatuswhiteboard
- usetargetmilestone

A user in the tweakparams group can access all existing parameters. New parameters can appear or obsolete parameters can disappear depending on the version of Bugzilla and on extensions being installed. The list of parameters returned by this method is not stable and will never be stable.

### Last Audit Time

Gets the most recent timestamp among all of the events recorded in the audit_log table.

**Request**

To get most recent audit timestamp for all classes:

```
GET /rest/last_audit_time
```

To get the the most recent audit timestamp for the `Bugzilla::Product` class:

```
GET /rest/last_audit_time?class=Bugzilla::Product
```

| name | type | description |
|---|---|---|
| class | array | The class names are defined as `Bugzilla::<class_name>`" such as Bugzilla:Product`` for products. |

**Response**

```
{
  "last_audit_time": "2014-09-23T18:03:38Z"
}
```

| name | type | description |
|---|---|---|
| last_audit_time | string | The maximum of the at_time from the audit_log. |

### Job Queue Status

Reports the status of the job queue.

**Request**

```
GET /rest/jobqueue_status
```

This method requires an authenticated user.

**Response**

```
{
  "total": 12,
  "errors": 0
}
```

| name | type | description |
|---|---|---|
| total | integer | The total number of jobs in the job queue. |
| errors | integer | The number of errors produced by jobs in the queue. |

---

This documentation undoubtedly has bugs; if you find some, please file them here.

### 5.1.7 Classifications

This part of the Bugzilla API allows you to deal with the available classifications. You will be able to get information about them as well as manipulate them.

#### Get Classification

Returns an object containing information about a set of classifications.

**Request**

To return information on a single classification using the ID or name:

```
GET /rest/classification/(id_or_name)
```

| name | type | description |
|---|---|---|
| **id_or_name** | mixed | An Integer classification ID or name. |

**Response**

```
{
  "classifications": [
    {
      "sort_key": 0,
      "description": "Unassigned to any classifications",
      "products": [
        {
          "id": 2,
          "name": "FoodReplicator",
          "description": "Software that controls a piece of hardware that will create␣
↪any food item through a voice interface."
        },
        {
          "description": "Silk, etc.",
          "name": "Spider Secretions",
          "id": 4
        }
      ],
      "id": 1,
      "name": "Unclassified"
    }
  ]
}
```

`classifications` (array) Each object is a classification that the user is authorized to see and has the following items:

| name | type | description |
|---|---|---|
| id | int | The ID of the classification. |
| name | string | The name of the classification. |
| description | string | The description of the classification. |
| sort_key | int | The value which determines the order the classification is sorted. |
| products | array | Products the user is authorized to access within the classification. The product object keys are described below. |

Product object:

| name | type | description |
|---|---|---|
| name | string | The name of the product. |
| id | int | The ID of the product. |
| description | string | The description of the product. |

---

This documentation undoubtedly has bugs; if you find some, please file them here.

### 5.1.8 Comments

#### Get Comments

This allows you to get data about comments, given a bug ID or comment ID.

**Request**

To get all comments for a particular bug using the bug ID or alias:

```
GET /rest/bug/(id_or_alias)/comment
```

To get a specific comment based on the comment ID:

```
GET /rest/bug/comment/(comment_id)
```

| name | type | description |
|---|---|---|
| **id_or_alias** | mixed | A single integer bug ID or alias. |
| **comment_id** | int | A single integer comment ID. |
| new_since | datetime | If specified, the method will only return comments *newer* than this time. This only affects comments returned from the `ids` argument. You will always be returned all comments you request in the `comment_ids` argument, even if they are older than this date. |

**Response**

```
{
  "bugs": {
    "35": {
      "comments": [
```

(continues on next page)

(continued from previous page)

```
        {
          "time": "2000-07-25T13:50:04Z",
          "text": "test bug to fix problem in removing from cc list.",
          "bug_id": 35,
          "count": 0,
          "attachment_id": null,
          "is_private": false,
          "tags": [],
          "creator": "user@bugzilla.org",
          "creation_time": "2000-07-25T13:50:04Z",
          "id": 75
        }
      ]
    }
  },
  "comments": {}
}
```

Two items are returned:

`bugs` This is used for bugs specified in `ids`. This is an object, where the keys are the numeric IDs of the bugs, and the value is a object with a single key, `comments`, which is an array of comments. (The format of comments is described below.)

Any individual bug will only be returned once, so if you specify an ID multiple times in `ids`, it will still only be returned once.

`comments` Each individual comment requested in `comment_ids` is returned here, in a object where the numeric comment ID is the key, and the value is the comment. (The format of comments is described below.)

A "comment" as described above is a object that contains the following items:

| name | type | description |
|---|---|---|
| id | int | The globally unique ID for the comment. |
| bug_id | int | The ID of the bug that this comment is on. |
| attachment_id | int | If the comment was made on an attachment, this will be the ID of that attachment. Otherwise it will be null. |
| count | int | The number of the comment local to the bug. The Description is 0, comments start with 1. |
| text | string | The body of the comment, including any special text (such as "this bug was marked as a duplicate of…"). |
| raw_text | string | The body of the comment without any special additional text. |
| creator | string | The login name of the comment's author. |
| time | datetime | The time (in Bugzilla's timezone) that the comment was added. |
| creation_time | datetime | This is exactly same as the `time` key. Use this field instead of `time` for consistency with other methods including *Get Bug* and *Get Attachment*.<br>For compatibility, `time` is still usable. However, please note that `time` may be deprecated and removed in a future release. |
| is_private | boolean | `true` if this comment is private (only visible to a certain group called the "insidergroup"), `false` otherwise. |
| is_markdown | boolean | `true` if this comment is markdown. `false` if this comment is plaintext. |

**Errors**

This method can throw all the same errors as *Get Bug*. In addition, it can also throw the following errors:

- 110 (Comment Is Private) You specified the id of a private comment in the "comment_ids" argument, and you are not in the "insider group" that can see private comments.
- 111 (Invalid Comment ID) You specified an id in the "comment_ids" argument that is invalid–either you specified something that wasn't a number, or there is no comment with that id.

## Create Comments

This allows you to add a comment to a bug in Bugzilla. All comments created via the API will be considered Markdown (specifically GitHub Flavored Markdown).

**Request**

To create a comment on a current bug.

```
POST /rest/bug/(id)/comment
```

```
{
  "ids" : [123,..],
  "comment" : "This is an additional comment",
  "is_private" : false,
  "is_markdown" : true
}
```

`ids` is optional in the data example above and can be used to specify adding a comment to more than one bug at the same time.

| name | type | description |
|---|---|---|
| **id** | int | The ID or alias of the bug to append a comment to. |
| ids | array | List of integer bug IDs to add the comment to. |
| **com-ment** | string | The comment to append to the bug. If this is empty or all whitespace, an error will be thrown saying that you did not set the `comment` parameter. |
| is_private | boolean | If set to true, the comment is private, otherwise it is assumed to be public. |
| is_markdown | boolean | If true, the comment will be rendered as markdown. (default: false) |
| work_time | dou-ble | Adds this many hours to the "Hours Worked" on the bug. If you are not in the time tracking group, this value will be ignored. |

**Response**

```
{
  "id" : 789
}
```

| name | type | description |
|---|---|---|
| id | int | ID of the newly-created comment. |

**Errors**

- 54 (Hours Worked Too Large) You specified a "work_time" larger than the maximum allowed value of "99999.99".

- 100 (Invalid Bug Alias) If you specified an alias and there is no bug with that alias.
- 101 (Invalid Bug ID) The id you specified doesn't exist in the database.
- 109 (Bug Edit Denied) You did not have the necessary rights to edit the bug.
- 113 (Can't Make Private Comments) You tried to add a private comment, but don't have the necessary rights.
- 114 (Comment Too Long) You tried to add a comment longer than the maximum allowed length (65,535 characters).
- 140 (Markdown Disabled) You tried to set the "is_markdown" flag to true but the Markdown feature is not enabled.

### Search Comment Tags

Searches for tags which contain the provided substring.

**Request**

To search for comment tags:

```
GET /rest/bug/comment/tags/(query)
```

Example:

```
GET /rest/bug/comment/tags/spa
```

| name | type | description |
|---|---|---|
| **query** | string | Only tags containing this substring will be returned. |
| limit | int | If provided will return no more than `limit` tags. Defaults to `10`. |

**Response**

```
[
  "spam"
]
```

An array of matching tags.

**Errors**

This method can throw all of the errors that *Get Bug* throws, plus:

- 125 (Comment Tagging Disabled) Comment tagging support is not available or enabled.

### Update Comment Tags

Adds or removes tags from a comment.

**Request**

To update the tags comments attached to a comment:

```
PUT /rest/bug/comment/(comment_id)/tags
```

Example:

```
{
  "comment_id" : 75,
  "add" : ["spam", "bad"]
}
```

| name | type | description |
|---|---|---|
| **comment_id** | int | The ID of the comment to update. |
| add | array | The tags to attach to the comment. |
| remove | array | The tags to detach from the comment. |

**Response**

```
[
  "bad",
  "spam"
]
```

An array of strings containing the comment's updated tags.

**Errors**

This method can throw all of the errors that *Get Bug* throws, plus:

- 125 (Comment Tagging Disabled) Comment tagging support is not available or enabled.
- 126 (Invalid Comment Tag) The comment tag provided was not valid (e.g. contains invalid characters).
- 127 (Comment Tag Too Short) The comment tag provided is shorter than the minimum length.
- 128 (Comment Tag Too Long) The comment tag provided is longer than the maximum length.

### Render Comment

Returns the HTML rendering of the provided comment text.

**Request**

```
POST /rest/bug/comment/render
```

Example:

```
{
  "id" : 2345,
  "text" : "This issue has been fixed in bug 1234."
}
```

| name | type | description |
|---|---|---|
| **text** | string | Comment text to render. |
| id | int | The ID of the bug to render the comment against. |

**Response**

```
{
  "html" : "This issue has been fixed in <a class=\"bz_bug_link
       bz_status_RESOLVED  bz_closed\" title=\"RESOLVED FIXED - some issue that was␣
↪fixed\" href=\"show_bug.cgi?id=1234\">bug 1234</a>."
]
```

| name | type | description |
|---|---|---|
| html | string | Text containing the HTML rendering. |

**Errors**

This method can throw all of the errors that *Get Bug* throws.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

### 5.1.9 Bug Fields

The Bugzilla API for getting details about bug fields.

#### Fields

Get information about valid bug fields, including the lists of legal values for each field.

**Request**

To get information about all fields:

```
GET /rest/field/bug
```

To get information related to a single field:

```
GET /rest/field/bug/(id_or_name)
```

| name | type | description |
|---|---|---|
| id_or_name | mixed | An integer field ID or string representing the field name. |

**Response**

```
{
  "fields": [
    {
      "display_name": "Priority",
      "name": "priority",
      "type": 2,
      "is_mandatory": false,
      "value_field": null,
      "values": [
        {
          "sortkey": 100,
          "sort_key": 100,
```

(continues on next page)

(continued from previous page)

```
          "visibility_values": [],
          "name": "P1"
        },
        {
          "sort_key": 200,
          "name": "P2",
          "visibility_values": [],
          "sortkey": 200
        },
        {
          "sort_key": 300,
          "visibility_values": [],
          "name": "P3",
          "sortkey": 300
        },
        {
          "sort_key": 400,
          "name": "P4",
          "visibility_values": [],
          "sortkey": 400
        },
        {
          "name": "P5",
          "visibility_values": [],
          "sort_key": 500,
          "sortkey": 500
        }
      ],
      "visibility_values": [],
      "visibility_field": null,
      "is_on_bug_entry": false,
      "is_custom": false,
      "id": 13
    }
  ]
}
```

`field` (array) Field objects each containing the following items:

| name | type | description |
|---|---|---|
| id | int | An integer ID uniquely identifying this field in this installation only. |
| type | int | The number of the fieldtype. The following values are defined: • `0` Field type unknown • `1` Single-line string field • `2` Single value field • `3` Multiple value field • `4` Multi-line text value • `5` Date field with time • `6` Bug ID field • `7` See Also field • `8` Keywords field • `9` Date field • `10` Integer field |
| is_custom | boolean | `true` when this is a custom field, `false` otherwise. |
| name | string | The internal name of this field. This is a unique identifier for this field. If this is not a custom field, then this name will be the same across all Bugzilla installations. |
| display_name | string | The name of the field, as it is shown in the user interface. |
| is_mandatory | boolean | `true` if the field must have a value when filing new bugs. Also, mandatory fields cannot have their value cleared when updating bugs. |
| is_on_bug_entry | boolean | For custom fields, this is `true` if the field is shown when you enter a new bug. For standard fields, this is currently always `false`, even if the field shows up when entering a bug. (To know whether or not a standard field is valid on bug entry, see *Create Bug*. |
| visibility_field | string | The name of a field that controls the visibility of this field in the user interface. This field only appears in the user interface when the named field is equal to one of the values is `visibility_values`. Can be null. |
| visibility_values | array | This field is only shown when `visibility_field` matches one of these string values. When `visibility_field` is null, then this is an empty array. |
| value_field | string | The name of the field that controls whether or not particular values of the field are shown in the user interface. Can be null. |
| values | array | Objects representing the legal values for select-type (drop-down and multiple-selection) fields. This is also populated for the `component`, `version`, `target_milestone`, and `keywords` fields, but not for the `product` field (you must use |

Value object:

| name | type | description |
|---|---|---|
| name | string | The actual value–this is what you would specify for this field in `create`, etc. |
| sort_key | int | Values, when displayed in a list, are sorted first by this integer and then secondly by their name. |
| visibility_values | array | If `value_field` is defined for this field, then this value is only shown if the `value_field` is set to one of the values listed in this array. Note that for per-product fields, `value_field` is set to `product` and `visibility_values` will reflect which product(s) this value appears in. |
| is_active | boolean | This value is defined only for certain product-specific fields such as version, target_milestone or component. When true, the value is active; otherwise the value is not active. |
| description | string | The description of the value. This item is only included for the `keywords` field. |
| is_open | boolean | For `bug_status` values, determines whether this status specifies that the bug is "open" (`true`) or "closed" (`false`). This item is only included for the `bug_status` field. |
| can_change_to | array | For `bug_status` values, this is an array of objects that determine which statuses you can transition to from this status. (This item is only included for the `bug_status` field.) Each object contains the following items:<br>• name: (string) The name of the new status<br>• comment_required: (boolean) `true` if a comment is required when you change a bug into this status using this transition. |

**Errors**

- 51 (Invalid Field Name or Id) You specified an invalid field name or id.

### Legal Values

**DEPRECATED** Use ‘’Fields” instead.

Tells you what values are allowed for a particular field.

**Request**

To get information on the values for a field based on field name:

```
GET /rest/field/bug/(field)/values
```

To get information based on field name and a specific product:

```
GET /rest/field/bug/(field)/(product_id)/values
```

| name | type | description |
|---|---|---|
| field | string | The name of the field you want information about. This should be the same as the name you would use in *Create Bug*, below. |
| prod-uct_id | int | If you’re picking a product-specific field, you have to specify the ID of the product you want the values for. |

**Response**

```
{
  "values": [
    "P1",
    "P2",
    "P3",
    "P4",
    "P5"
  ]
}
```

| name | type | description |
|---|---|---|
| values | array | The legal values for this field. The values will be sorted as they normally would be in Bugzilla. |

**Errors**

- 106 (Invalid Product) You were required to specify a product, and either you didn’t, or you specified an invalid product (or a product that you can’t access).
- 108 (Invalid Field Name) You specified a field that doesn’t exist or isn’t a drop-down field.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

### 5.1.10 Groups

The API for creating, changing, and getting information about groups.

#### Create Group

This allows you to create a new group in Bugzilla. You must be authenticated and be in the *creategroups* group to perform this action.

**Request**

```
POST /rest/group
```

```
{
  "name" : "secret-group",
  "description" : "Too secret for you!",
  "is_active" : true
}
```

Some params must be set, or an error will be thrown. The required params are marked in **bold**.

| name | type | description |
|---|---|---|
| **name** | string | A short name for this group. Must be unique. This is not usually displayed in the user interface, except in a few places. |
| **description** | string | A human-readable name for this group. Should be relatively short. This is what will normally appear in the UI as the name of the group. |
| user_regexp | string | A regular expression. Any user whose Bugzilla username matches this regular expression will automatically be granted membership in this group. |
| is_active | boolean | `true` if new group can be used for bugs, `false` if this is a group that will only contain users and no bugs will be restricted to it. |
| icon_url | string | A URL pointing to a small icon used to identify the group. This icon will show up next to users' names in various parts of Bugzilla if they are in this group. |

**Response**

```
{
  "id": 22
}
```

| name | type | description |
|---|---|---|
| id | int | ID of the newly-created group. |

**Errors**

- 800 (Empty Group Name) You must specify a value for the "name" field.
- 801 (Group Exists) There is already another group with the same "name".
- 802 (Group Missing Description) You must specify a value for the "description" field.
- 803 (Group Regexp Invalid) You specified an invalid regular expression in the "user_regexp" field.

### Update Group

This allows you to update a group in Bugzilla. You must be authenticated and be in the *creategroups* group to perform this action.

**Request**

To update a group using the group ID or name:

```
PUT /rest/group/(id_or_name)
```

```
{
  "name" : "secret-group",
  "description" : "Too secret for you! (updated description)",
  "is_active" : false
}
```

You can edit a single group by passing the ID or name of the group in the URL. To edit more than one group, you can specify addition IDs or group names using the `ids` or `names` parameters respectively.

One of the below must be specified.

| name | type | description |
|---|---|---|
| **id_or_name** | mixed | Integer group or name. |
| **ids** | array | IDs of groups to update. |
| **names** | array | Names of groups to update. |

The following parameters specify the new values you want to set for the group(s) you are updating.

| name | type | description |
|---|---|---|
| name | string | A new name for the groups. If you try to set this while updating more than one group, an error will occur, as group names must be unique. |
| de-scrip-tion | string | A new description for the groups. This is what will appear in the UI as the name of the groups. |
| user_regexp | string | A new regular expression for email. Will automatically grant membership to these groups to anyone with an email address that matches this Perl regular expression. |
| is_active | boolean | Set if groups are active and eligible to be used for bugs. `true` if bugs can be restricted to this group, `false` otherwise. |
| icon_url | string | A URL pointing to an icon that will appear next to the name of users who are in this group. |

**Response**

```
{
  "groups": [
    {
      "changes": {
        "description": {
          "added": "Too secret for you! (updated description)",
          "removed": "Too secret for you!"
        },
        "is_active": {
          "removed": "1",
```

(continues on next page)

(continued from previous page)

```
          "added": "0"
        }
      },
      "id": "22"
    }
  ]
}
```

`groups` (array) Group change objects, each containing the following items:

| name | type | description |
|---|---|---|
| id | int | The ID of the group that was updated. |
| changes | object | The changes that were actually done on this group. The keys are the names of the fields that were changed, and the values are an object with two items:<br>• added: (string) The values that were added to this field, possibly a comma-and-space-separated list if multiple values were added.<br>• removed: (string) The values that were removed from this field, possibly a comma-and-space-separated list if multiple values were removed. |

**Errors**

The same as *Create Group*.

### Get Group

Returns information about Bugzilla groups.

**Request**

To return information about a specific group ID or name:

```
GET /rest/group/(id_or_name)
```

You can also return information about more than one specific group by using the following in your query string:

```
GET /rest/group?ids=1&ids=2&ids=3
GET /group?names=ProductOne&names=Product2
```

If neither IDs nor names are passed, and you are in the creategroups or editusers group, then all groups will be retrieved. Otherwise, only groups that you have bless privileges for will be returned.

| name | type | description |
|---|---|---|
| id_or_name | mixed | Integer group ID or name. |
| ids | array | Integer IDs of groups. |
| names | array | Names of groups. |
| membership | boolean | Set to 1 then a list of members of the passed groups names and IDs will be returned. |

**Response**

```
{
  "groups": [
    {
      "membership": [
        {
          "real_name": "Bugzilla User",
          "nick": "user",
          "can_login": true,
          "name": "user@bugzilla.org",
          "login_denied_text": "",
          "id": 85,
          "email_enabled": false,
          "email": "user@bugzilla.org"
        },
      ],
      "is_active": true,
      "description": "Test Group",
      "user_regexp": "",
      "is_bug_group": true,
      "name": "TestGroup",
      "id": 9
    }
  ]
}
```

If the user is a member of the *creategroups* group they will receive information about all groups or groups matching the criteria that they passed. You have to be in the creategroups group unless you're requesting membership information.

If the user is not a member of the *creategroups* group, but they are in the "editusers" group or have bless privileges to the groups they require membership information for, the is_active, is_bug_group and user_regexp values are not supplied.

The return value will be an object containing group names as the keys; each value will be an object that describes the group and has the following items:

| name | type | description |
|---|---|---|
| id | int | The unique integer ID that Bugzilla uses to identify this group. Even if the name of the group changes, this ID will stay the same. |
| name | string | The name of the group. |
| description | string | The description of the group. |
| is_bug_group | int | Whether this group is to be used for bug reports or is only administrative specific. |
| user_regexp | string | A regular expression that allows users to be added to this group if their login matches. |
| is_active | int | Whether this group is currently active or not. |
| users | array | User objects that are members of this group; only returned if the user sets the `membership` parameter to 1. Each user object has the items describe in the User object below. |

User object:

| name | type | description |
|---|---|---|
| id | int | The ID of the user. |
| real_name | string | The actual name of the user. |
| nick | string | The user's nickname. Currently this is extracted from the real_name, name or email field. |
| email | string | The email address of the user. |
| name | string | The login name of the user. Note that in some situations this is different than their email. |
| can_login | boolean | A boolean value to indicate if the user can login into Bugzilla. |
| email_enabled | boolean | A boolean value to indicate if bug-related mail will be sent to the user or not. |
| disabled_text | string | A text field that holds the reason for disabling a user from logging into Bugzilla. If empty, then the user account is enabled; otherwise it is disabled/closed. |

**Errors**

- 51 (Invalid Object) A non existing group name was passed to the function, as a result no group object existed for that invalid name.
- 805 (Cannot view groups) Logged-in users are not authorized to edit Bugzilla groups as they are not members of the creategroups group in Bugzilla, or they are not authorized to access group member's information as they are not members of the "editusers" group or can bless the group.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

### 5.1.11 Products

This part of the Bugzilla API allows you to list the available products and get information about them.

#### List Products

Returns a list of the IDs of the products the user can search on.

**Request**

To get a list of product IDs a user can select such as for querying bugs:

```
GET /rest/product_selectable
```

To get a list of product IDs a user can enter a bug against:

```
GET /rest/product_enterable
```

To get a list of product IDs a user can search or enter bugs against.

```
GET /rest/product_accessible
```

**Response**

```
{
  "ids": [
    "2",
    "3",
    "19",
    "1",
    "4"
  ]
}
```

| name | type | description |
|---|---|---|
| ids | array | List of integer product IDs. |

### Get Product

Returns a list of information about the products passed to it.

**Request**

To return information about a specific type of products such as `accessible`, `selectable`, or `enterable`:

```
GET /rest/product?type=accessible
```

To return information about a specific product by `id` or `name`:

```
GET /rest/product/(id_or_name)
```

You can also return information about more than one product by using the following parameters in your query string:

```
GET /rest/product?ids=1&ids=2&ids=3
GET /rest/product?names=ProductOne&names=Product2
```

| name | type | description |
|---|---|---|
| id_or_name | mixed | Integer product ID or product name. |
| ids | array | Product IDs |
| names | array | Product names |
| type | string | The group of products to return. Valid values are `accessible` (default), `selectable`, and `enterable`. `type` can be a single value or an array of values if more than one group is needed with duplicates removed. |

**Response**

```
{
  "products": [
    {
      "id": 1,
      "default_bug_type": "defect",
      "default_milestone": "---",
      "components": [
        {
          "is_active": true,
          "default_assigned_to": "admin@bugzilla.org",
          "default_bug_type": "defect",
          "id": 1,
          "sort_key": 0,
          "name": "TestComponent",
          "flag_types": {
            "bug": [
              {
                "is_active": true,
                "grant_group": null,
                "cc_list": "",
                "is_requestable": true,
                "id": 3,
                "is_multiplicable": true,
                "name": "needinfo",
                "request_group": null,
                "is_requesteeble": true,
                "sort_key": 0,
                "description": "needinfo"
              }
            ],
            "attachment": [
              {
                "description": "Review",
                "is_multiplicable": true,
                "name": "review",
                "is_requesteeble": true,
                "request_group": null,
                "sort_key": 0,
                "cc_list": "",
                "grant_group": null,
                "is_requestable": true,
                "id": 2,
                "is_active": true
              }
            ]
          },
          "default_qa_contact": "",
          "triage_owner": "",
          "description": "This is a test component."
        }
      ],
      "is_active": true,
      "classification": "Unclassified",
```

(continues on next page)

(continued from previous page)

```
      "versions": [
        {
          "id": 1,
          "name": "unspecified",
          "is_active": true,
          "sort_key": 0
        }
      ],
      "description": "This is a test product.",
      "has_unconfirmed": true,
      "milestones": [
        {
          "name": "---",
          "is_active": true,
          "sort_key": 0,
          "id": 1
        }
      ],
      "name": "TestProduct"
    }
  ]
}
```

`products` (array) Each product object has the following items:

| name | type | description |
|---|---|---|
| id | int | An integer ID uniquely identifying the product in this installation only. |
| name | string | The name of the product. This is a unique identifier for the product. |
| description | string | A description of the product, which may contain HTML. |
| is_active | boolean | A boolean indicating if the product is active. |
| de-fault_bug_type | string | The default type for bugs filed under this product. |
| de-fault_milestone | string | The name of the default milestone for the product. |
| has_unconfirmed | boolean | Indicates whether the UNCONFIRMED bug status is available for this product. |
| classification | string | The classification name for the product. |
| components | array | Each component object has the items described in the Component object below. |
| versions | array | Each object describes a version, and has the following items: `name`, `sort_key` and `is_active`. |
| milestones | array | Each object describes a milestone, and has the following items: `name`, `sort_key` and `is_active`. |

If the user tries to access a product that is not in the list of accessible products for the user, or a product that does not exist, that is silently ignored, and no information about that product is returned.

Component object:

| name | type | description |
|---|---|---|
| id | int | An integer ID uniquely identifying the component in this installation only. |
| name | string | The name of the component. This is a unique identifier for this component. |
| description | string | A description of the component, which may contain HTML. |
| default_assigned_to | string | The login name of the user to whom new bugs will be assigned by default. |
| default_bug_type | string | The default type for bugs filed under this component. |
| default_qa_contact | string | The login name of the user who will be set as the QA Contact for new bugs by default. Empty string if the QA contact is not defined. |
| triage_owner | string | The login name of the user who is named as the Triage Owner of the component. Empty string if the Triage Owner is not defined. |
| sort_key | int | Components, when displayed in a list, are sorted first by this integer and then secondly by their name. |
| is_active | boolean | A boolean indicating if the component is active. Inactive components are not enabled for new bugs. |
| flag_types | object | An object containing two items `bug` and `attachment` that each contains an array of objects, where each describes a flagtype. The flagtype items are described in the Flagtype object below. |

Flagtype object:

| name | type | description |
|---|---|---|
| id | int | Returns the ID of the flagtype. |
| name | string | Returns the name of the flagtype. |
| description | string | Returns the description of the flagtype. |
| cc_list | string | Returns the concatenated CC list for the flagtype, as a single string. |
| sort_key | int | Returns the sortkey of the flagtype. |
| is_active | boolean | Returns whether the flagtype is active or disabled. Flags being in a disabled flagtype are not deleted. It only prevents you from adding new flags to it. |
| is_requestable | boolean | Returns whether you can request for the given flagtype (i.e. whether the '?' flag is available or not). |
| is_requesteeble | boolean | Returns whether you can ask someone specifically or not. |
| is_multiplicable | boolean | Returns whether you can have more than one flag for the given flagtype in a given bug/attachment. |
| grant_group | int | the group ID that is allowed to grant/deny flags of this type. If the item is not included all users are allowed to grant/deny this flagtype. |
| request_group | int | The group ID that is allowed to request the flag if the flag is of the type requestable. If the item is not included all users are allowed request this flagtype. |

To return information about components in products, you can use the `.` property accesssor in your request:

```
/rest/product?type=enterable&include_fields=id,name,components.name,components.id,
↪components.is_active,components.description
```

### Create Product

This allows you to create a new product in Bugzilla.

**Request**

```
POST /rest/product
```

```
{
  "name" : "AnotherProduct",
  "description" : "Another Product",
  "classification" : "Unclassified",
  "is_open" : false,
  "has_unconfirmed" : false,
  "version" : "unspecified"
}
```

Some params must be set, or an error will be thrown. The required params are marked in bold.

| name | type | description |
|---|---|---|
| **name** | string | The name of this product. Must be globally unique within Bugzilla. |
| **description** | string | A description for this product. Allows some simple HTML. |
| **version** | string | The default version for this product. |
| has_unconfirmed | boolean | Allow the UNCONFIRMED status to be set on bugs in this product. Default: true. |
| classification | string | The name of the Classification which contains this product. |
| de-fault_bug_type | string | The default type for bugs filed under this product. Each component can override this value. |
| de-fault_milestone | string | The default milestone for this product. Default '—'. |
| is_open | boolean | `true` if the product is currently allowing bugs to be entered into it. Default: `true`. |
| create_series | boolean | `true` if you want series for New Charts to be created for this new product. Default: `true`. |

**Response**

```
{
  "id": 20
}
```

Returns an object with the following items:

| name | type | description |
|---|---|---|
| id | int | ID of the newly-filed product. |

**Errors**

- 51 (Classification does not exist) You must specify an existing classification name.
- 700 (Product blank name) You must specify a non-blank name for this product.
- 701 (Product name too long) The name specified for this product was longer than the maximum allowed length.
- 702 (Product name already exists) You specified the name of a product that already exists. (Product names must be globally unique in Bugzilla.)
- 703 (Product must have description) You must specify a description for this product.

- 704 (Product must have version) You must specify a version for this product.

### Update Product

This allows you to update a product in Bugzilla.

**Request**

```
PUT /rest/product/(id_or_name)
```

You can edit a single product by passing the ID or name of the product in the URL. To edit more than one product, you can specify addition IDs or product names using the `ids` or `names` parameters respectively.

```
{
  "ids" : [123],
  "name" : "BarName",
  "has_unconfirmed" : false
}
```

One of the below must be specified.

| name | type | description |
|---|---|---|
| **id_or_name** | mixed | Integer product ID or name. |
| **ids** | array | Numeric IDs of the products that you wish to update. |
| **names** | array | Names of the products that you wish to update. |

The following parameters specify the new values you want to set for the product(s) you are updating.

| name | type | description |
|---|---|---|
| name | string | A new name for this product. If you try to set this while updating more than one product, an error will occur, as product names must be unique. |
| default_bug_type | string | The default type for bugs filed under this product. Each component can override this value. |
| default_milestone | string | When a new bug is filed, what milestone does it get by default if the user does not choose one? Must represent a milestone that is valid for this product. |
| description | string | Update the long description for these products to this value. |
| has_unconfirmed | boolean | Allow the UNCONFIRMED status to be set on bugs in products. |
| is_open | boolean | `true` if the product is currently allowing bugs to be entered into it, `false` otherwise. |

**Response**

```
{
   "products" : [
      {
         "id" : 123,
         "changes" : {
            "name" : {
               "removed" : "FooName",
               "added" : "BarName"
            },
            "has_unconfirmed" : {
```

(continues on next page)

(continued from previous page)

```
                "removed" : "1",
                "added" : "0"
            }
        }
    }
  ]
}
```

`products` (array) Product change objects containing the following items:

| name | type | description |
|---|---|---|
| id | int | The ID of the product that was updated. |
| changes | object | The changes that were actually done on this product. The keys are the names of the fields that were changed, and the values are an object with two items:<br>• added: (string) The value that this field was changed to.<br>• removed: (string) The value that was previously set in this field. |

Booleans will be represented with the strings '1' and '0' for changed values as they are stored as strings in the database currently.

**Errors**

- 700 (Product blank name) You must specify a non-blank name for this product.
- 701 (Product name too long) The name specified for this product was longer than the maximum allowed length.
- 702 (Product name already exists) You specified the name of a product that already exists. (Product names must be globally unique in Bugzilla.)
- 703 (Product must have description) You must specify a description for this product.
- 705 (Product must define a default milestone) You must define a default milestone.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

## 5.1.12 Users

This part of the Bugzilla API allows you to create user accounts, get information about user accounts and to log in or out using an existing account.

## Login

Logging in with a username and password is required for many Bugzilla installations, in order to search for private bugs, post new bugs, etc. This method allows you to retrieve a token that can be used as authentication for subsequent API calls. Otherwise you will need to pass your `login` and `password` with each call.

This method will be going away in the future in favor of using *API keys*.

**Request**

```
GET /rest/login?login=foo@example.com&password=toosecrettoshow
```

| name | type | description |
|---|---|---|
| **login** | string | The user's login name. |
| **password** | string | The user's password. |

**Response**

```
{
  "token": "786-OLaWfBisMY",
  "id": 786
}
```

| name | type | description |
|---|---|---|
| id | int | Numeric ID of the user that was logged in. |
| to-ken | string | Token which can be passed in the parameters as authentication in other calls. The token can be sent along with any future requests to the webservice, for the duration of the session, i.e. til *Logout* is called. |

**Errors**

- 300 (Invalid Username or Password) The username does not exist, or the password is wrong.
- 301 (Login Disabled) The ability to login with this account has been disabled. A reason may be specified with the error.
- 305 (New Password Required) The current password is correct, but the user is asked to change their password.
- 50 (Param Required) A login or password parameter was not provided.

## Logout

Log out the user. Basically it invalidates the token provided so it cannot be re-used. Does nothing if the token is not in use. Will also clear any existing authentication cookies the client may still have stored.

**Request**

```
GET /rest/logout?token=1234-VWvO51X69r
```

| name | type | description |
|---|---|---|
| token | string | The user's token used for authentication. |

### Valid Login

This method will verify whether a client's cookies or current login token is still valid or have expired. A valid username that matches must be provided as well.

**Request**

```
GET /rest/valid_login?login=foo@example.com&token=1234-VWvO51X69r
```

| name | type | description |
|---|---|---|
| **login** | string | The login name that matches the provided cookies or token. |
| token | string | Persistent login token currently being used for authentication. |

**Response**

Returns true/false depending on if the current token is valid for the provided username.

### Create User

Creates a user account directly in Bugzilla, password and all. Instead of this, you should use **Offer Account by Email** when possible because that makes sure that the email address specified can actually receive an email. This function does not check that. You must be authenticated and be in the *editusers* group to perform this action.

**Request**

```
POST /rest/user
```

```
{
  "email" : "user@bugzilla.org",
  "full_name" : "Test User",
  "password" : "K16ldRr922I1"
}
```

| name | type | description |
|---|---|---|
| **email** | string | The email address for the new user. |
| full_name | string | The user's full name. Will be set to empty if not specified. |
| password | string | The password for the new user account, in plain text. It will be stripped of leading and trailing whitespace. If blank or not specified, the new created account will exist in Bugzilla but will not be allowed to log in using DB authentication until a password is set either by the user (through resetting their password) or by the administrator. |

**Response**

```
{
  "id": 58707
}
```

| name | type | description |
|---|---|---|
| id | int | The numeric ID of the user that was created. |

**Errors**

- 502 (Password Too Short) The password specified is too short. (Usually, this means the password is under three characters.)

### Update User

Updates an existing user account in Bugzilla. You must be authenticated and be in the *editusers* group to perform this action.

**Request**

```
PUT /rest/user/(id_or_name)
```

You can edit a single user by passing the ID or login name of the user in the URL. To edit more than one user, you can specify addition IDs or login names using the `ids` or `names` parameters respectively.

| name | type | description |
|---|---|---|
| **id_or_name** | mixed | Either the ID or the login name of the user to update. |
| **ids** | array | Additional IDs of users to update. |
| **names** | array | Additional login names of users to update. |
| full_name | string | The new name of the user. |
| email | string | The email of the user. Note that email used to login to Bugzilla. Also note that you can only update one user at a time when changing the login name / email. (An error will be thrown if you try to update this field for multiple users at once.) |
| password | string | The password of the user. |
| email_enabled | boolean | A boolean value to enable/disable sending bug-related mail to the user. |
| login_denied_text | string | A text field that holds the reason for disabling a user from logging into Bugzilla. If empty, then the user account is enabled; otherwise it is disabled/closed. |
| groups | object | These specify the groups that this user is directly a member of. To set these, you should pass an object as the value. The object's items are described in the Groups update objects below. |
| bless_groups | object | This is the same as groups but affects what groups a user has direct membership to bless that group. It takes the same inputs as groups. |

Groups and bless groups update object:

| name | type | description |
|---|---|---|
| add | array | The group IDs or group names that the user should be added to. |
| remove | array | The group IDs or group names that the user should be removed from. |
| set | array | Integers or strings which are an exact set of group IDs and group names that the user should be a member of. This does not remove groups from the user when the person making the change does not have the bless privilege for the group. |

If you specify `set`, then `add` and `remove` will be ignored. A group in both the `add` and `remove` list will be added. Specifying a group that the user making the change does not have bless rights will generate an error.

**Response**

- users: (array) List of user change objects with the following items:

| name | type | description |
|---|---|---|
| id | int | The ID of the user that was updated. |
| changes | object | The changes that were actually done on this user. The keys are the names of the fields that were changed, and the values are an object with two items:<br>• added: (string) The values that were added to this field, possibly a comma-and-space-separated list if multiple values were added.<br>• removed: (string) The values that were removed from this field, possibly a comma-and-space-separated list if multiple values were removed. |

**Errors**

- 51 (Bad Login Name) You passed an invalid login name in the “names” array.
- 304 (Authorization Required) Logged-in users are not authorized to edit other users.

## Get User

Gets information about user accounts in Bugzilla.

**Request**

To get information about a single user in Bugzilla:

```
GET /rest/user/(id_or_name)
```

To get multiple users by name or ID:

```
GET /rest/user?names=foo@bar.com&name=test@bugzilla.org
GET /rest/user?ids=123&ids=321
```

To get user matching a search string:

```
GET /rest/user?match=foo
```

To get user by using an integer ID value or by using `match`, you must be authenticated.

| name | type | description |
|---|---|---|
| id_or_name | mixed | An integer user ID or login name of the user. |
| ids | array | Integer user IDs. Logged=out users cannot pass this parameter to this function. If they try, they will get an error. Logged=in users will get an error if they specify the ID of a user they cannot see. |
| names | array | Login names. |
| match | array | This works just like "user matching" in Bugzilla itself. Users will be returned whose real name or login name contains any one of the specified strings. Users that you cannot see will not be included in the returned list.<br>Most installations have a limit on how many matches are returned for each string; the default is 1000 but can be changed by the Bugzilla administrator.<br>Logged-out users cannot use this argument, and an error will be thrown if they try. (This is to make it harder for spammers to harvest email addresses from Bugzilla, and also to enforce the user visibility restrictions that are implemented on some Bugzillas.) |
| limit | int | Limit the number of users matched by the `match` parameter. If the value is greater than the system limit, the system limit will be used. This parameter is only valid when using the `match` parameter. |
| group_ids | array | Numeric IDs for groups that a user can be in. |
| groups | array | Names of groups that a user can be in. If `group_ids` or `groups` are specified, they limit the return value to users who are in *any* of the groups specified. |
| include_disabled | boolean | By default, when using the `match` parameter, disabled users are excluded from the returned results unless their full username is identical to the match string. Setting `include_disabled` to `true` will include disabled users in the returned results even if their username doesn't fully match the input string. |

**Response**

- users: (array) Each object describes a user and has the following items:

| name | type | description |
|---|---|---|
| id | int | The unique integer ID that Bugzilla uses to represent this user. Even if the user's login name changes, this will not change. |
| real_name | string | The actual name of the user. May be blank. |
| nick | string | The user's nickname. Currently this is extracted from the real_name, name or email field. |
| email | string | The email address of the user. |
| name | string | The login name of the user. Note that in some situations this is different than their email. |
| can_login | boolean | A boolean value to indicate if the user can login into Bugzilla. |
| email_enabled | boolean | A boolean value to indicate if bug-related mail will be sent to the user or not. |
| login_denied_text | string | A text field that holds the reason for disabling a user from logging into Bugzilla. If empty then the user account is enabled; otherwise it is disabled/closed. |
| groups | array | Groups the user is a member of. If the currently logged in user is querying their own account or is a member of the 'editusers' group, the array will contain all the groups that the user is a member of. Otherwise, the array will only contain groups that the logged in user can bless. Each object describes the group and contains the items described in the Group object below. |
| saved_searches | array | User's saved searches, each having the following Search object items described below. |
| saved_reports | array | User's saved reports, each having the following Search object items described below. |

Group object:

| name | type | description |
|---|---|---|
| id | int | The group ID |
| name | string | The name of the group |
| description | string | The description for the group |

Search object:

| name | type | description |
|---|---|---|
| id | int | An integer ID uniquely identifying the saved report. |
| name | string | The name of the saved report. |
| query | string | The CGI parameters for the saved report. |

If you are not authenticated when you call this function, you will only be returned the `id`, `name`, `real_name` and `nick` items. If you are authenticated and not in 'editusers' group, you will only be returned the `id`, `name`, `real_name`, `nick`, `email`, `can_login` and `groups` items. The groups returned are filtered based on your permission to bless each group. The `saved_searches` and `saved_reports` items are only returned if you are querying your own account, even if you are in the editusers group.

**Errors**

- 51 (Bad Login Name or Group ID) You passed an invalid login name in the "names" array or a bad group ID in the "group_ids" argument.
- 52 (Invalid Parameter) The value used must be an integer greater than zero.
- 304 (Authorization Required) You are logged in, but you are not authorized to see one of the users you wanted to get information about by user id.
- 505 (User Access By Id or User-Matching Denied) Logged-out users cannot use the "ids" or "match" arguments to this function.
- 804 (Invalid Group Name) You passed a group name in the "groups" argument which either does not exist or you do not belong to it.

## Who Am I

Allows for validating a user's API key, token, or username and password. If successfully authenticated, it returns simple information about the logged in user.

**Request**

```
GET /rest/whoami
```

**Response**

```
{
  "id" : "1234",
  "name" : "user@bugzilla.org",
  "real_name" : "Test User",
  "nick" : "user"
}
```

| name | type | description |
|---|---|---|
| id | int | The unique integer ID that Bugzilla uses to represent this user. Even if the user's login name changes, this will not change. |
| real_name | string | The actual name of the user. May be blank. |
| nick | string | The user's nickname. Currently this is extracted from the real_name, name or email field. |
| name | string | string The login name of the user. |

---

This documentation undoubtedly has bugs; if you find some, please file them here.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

---

This documentation undoubtedly has bugs; if you find some, please file them here.

---

This documentation undoubtedly has bugs; if you find some, please file them here.